

STABLAB Documentation for KdV

BLAKE BARKER

January 22, 2014

Contents

1	Introduction	4
2	Study of single wave	5
2.1	bound_numer.m	5
2.2	bound_numer_unstable.m	9
2.3	bound_sub_integrals.m	13
2.4	bound_theta1.m.m	16
2.5	bound_xi.m	18
2.6	bound_xi_der.m	21
2.7	bound_xi_der_n1.m	25
2.8	bound_xi_n1.m	28
2.9	cf_biv_cheby.m	30
2.10	cf_eval.m	32
2.11	distinct.m	33
2.12	instability.m	34
2.13	integrand_numer.m	37
2.14	kappa_of_k.m	41
2.15	lambda_xi.m	42
2.16	lower_bound.m	44
2.17	lower_bound_unstable.m	47
2.18	numer.m	50
2.19	simplicity.m	54
2.20	stability.m	59
2.21	theta_vec.m	66
2.22	theta_vec_z.m	70
2.23	verify_data_n0.m	73
2.24	verify_data_n1.m	79
2.25	xi_der_q_psi.m	82
2.26	xi_q_psi.m	85
3	Full study	89
3.1	N_nodes.m	89
3.2	agm.m	90
3.3	bound_numer.m	91

3.4	bound_numer_unstable.m	96
3.5	bound_sub_integrals.m	100
3.6	bound_theta1.m.m	103
3.7	bound_xi.m	106
3.8	bound_xi_der.m	109
3.9	cf_biv_cheby.m	112
3.10	cf_eval.m	114
3.11	cf_eval_theta.m	115
3.12	cheby_eval.m	116
3.13	cheby_taylor.m	117
3.14	check_alpha_distinct.m	119
3.15	create_tableA.m	120
3.16	distinct.m	122
3.17	divide_interpolation.m	123
3.18	driver.m	125
3.19	driver_instability_upper.m	127
3.20	driver_stability_n0.m	131
3.21	driver_stability_n1.m	135
3.22	driver_strict_lower.m	141
3.23	driver_strict_upper_taylor.m	143
3.24	elliptic_integral.m	146
3.25	instability_interpolation.m	149
3.26	integrand_numer.m	152
3.27	interpolation_2d.m	156
3.28	kappa_lemma.m	162
3.29	kappa_of_k.m	165
3.30	lambda_xi.m	166
3.31	lemma_qk.m	168
3.32	local_startup.m	170
3.33	local_startup_batch.m	170
3.34	lower_bound.m	171
3.35	lower_bound_unstable.m	173
3.36	lower_instability_interpolation.m	176
3.37	numer.m	180
3.38	save_figure.m	184
3.39	simplicity.m	185
3.40	simplicity_aux.m	187
3.41	strict_taylor.m	191
3.42	strict_transition.m	192
3.43	strict_transition_lower.m	197
3.44	theta_vec.m	202
3.45	upper_pinpoint.m	206
3.46	verify_instability_lower.m	209
3.47	verify_instability_upper.m	211
3.48	verify_stability_n0_strict.m	214
3.49	verify_stability_n1.m	222

3.50	verify_stability_n1_strict.m	227
3.51	verify_stability_single.m	233
3.52	weierstrass_eta1.m	235
3.53	weierstrass_invariants.m	237
3.54	weierstrass_p.m	238
3.55	weierstrass_p_der.m	239
3.56	weierstrass_sigma.m	242
3.57	weierstrass_zeta.m	245
3.58	xi_der_q_psi.m	248
3.59	xi_q_psi.m	251

Chapter 1

Introduction

We document the STABLAB code used to study the stability of traveling-wave solutions of the Kuramoto-Sivashinsky equation in the Korteweg-de Vries limit.

Chapter 2

Study of single wave

2.1 bound_numer.m

```
function [M_psi,M_x,M_q] = bound_numer(dm,rho_psi,rho_x,rho_q,a_q,b_q,a_psi,b_psi,ntilde)

%
% constants
%

min_abs_q = (a_q+b_q)/2 -(b_q-a_q)*(rho_q+1/rho_q)/4;

%
% bound for interpolation in psi
%

% top of ellipse  $E_{\rho_\psi}$ 
max_imag_x = 0;
max_real_psi = (1+(rho_psi+1/rho_psi)/2)/2;
max_abs_q = sup(real(b_q));
prodq = abs(log(a_q)/2);

max_xi = bound_xi(a_psi,b_psi,rho_psi,a_q,b_q,1,ntilde);
max_xi_der = bound_xi_der...
    (max_abs_q,min_abs_q,a_q,b_q,max_real_psi,rho_q,rho_psi,a_psi,b_psi,ntilde);
```

```
M_psi = 2*local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);
```

```
%
% bound for interpolation in x -----
%
```

```
% top of ellipse  $E_{\rho_\beta}$ 
max_imag_x = (rho_x-1/rho_x)/2;
max_real_psi = 1;
% max_abs_q the same
% prodq the same
% max_xi the same
% max_xi_der the same
```

```
M_x = local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);
```

```
%
% bound for interpolation in q -----
%
```

```
% top of ellipse  $E_{\rho_\beta}$ 
max_imag_x = 0;
max_real_psi = 1;
max_abs_q = (a_q+b_q)/2 + ((b_q-a_q)/4)*(rho_q+1/rho_q);
```

```
max_xi = bound_xi(a_psi,b_psi,rho_psi,a_q,b_q,1,ntilde);
max_xi_der = bound_xi_der...
    (max_abs_q,min_abs_q,a_q,b_q,max_real_psi,rho_q,rho_psi,a_psi,b_psi,ntilde);
```

```
min_abs_q = (a_q+b_q)/2 - (b_q-a_q)*(rho_q+1/rho_q)/4;
prodq = abs(log(min_abs_q)/2);
```

```
M_q = 2*local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);
```

```
%-----
% local_bounds
```

```

%-----

function out = local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,xi,xi_der,prodq)

pie = nm('pi');
prod = pie/2;

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
n0 = temp(1);
n1 = temp(2); % first derivative
n2 = temp(3); % second derivative
n3 = temp(4); % third derivative
n4 = temp(5);

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
max_real_psi = 0;
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
d1 = temp(2); % first derivative
d2 = temp(3); % second derivative
d3 = temp(4); % third derivative
d4 = temp(5);

% bound on w(x), the conjugate of w(x), and their derivatives
% on the ellipse  $E_{\rho_x}$ 

% w(x)
w0 = n0^2/dm^2;

% w'(x)
w1 = 2*prod*( n0*n1/dm^2 + w0*d1/dm );

% w''(x)
w2 = 4*prod^2*(n1^2/dm^2+n0*n2/dm^2+2*n0*n1*d1/dm^3+w1*d1/dm+w0*d2/dm+w0*d1^2/dm^2);

% w'''(x)
w3 = 8*prod^3*( 2*n1*n2/dm^2+2*n1^2*d1/dm^3+n1*n2/dm^2+n0*n3/dm^2+2*n0*n2*d1/dm^3+...

```



```

2*n1^2*d1/dm^3+2*n0*n2*d1/dm^3+2*n0*n1*d2/dm^3+6*n0*n1*d1^2/dm^4+w2*d1/dm-
+ w1*d2/dm+...
w1*d1^2/dm^2+w1*d2/dm+ w0*d3/dm+w0*d1*d2/dm^2+w1*d1^2/dm^2+2*w0*d1*d2/dm^2+...
2*w0*d1^3/dm^3);

```

```

% bound on derivative of w(x) with respect to psi
w0_psi = 2*prodq*(n0*n1/dm^2);

```

```

% derivative of w'(x) with respect to psi
w1_psi = 2*prodq^2*(n0*n2/dm^2+n1^2/dm^2+w0_psi*d1/dm);

```

```

% bound on derivative of w''(x) with respect to psi
% —w''(x)— < 4*prodq^2*( (2*n0*n1*d1)/dm^3+ (n1^2+n0*n2+w0*d1^2)/dm^2-
+ (w1*d1+ w0*d2)/dm );
w2_psi = 4*prodq^3*( (2*n1^2*d1+2*n0*n2*d1+2*n0*n1*d2)/dm^3 + ...
(2*n1*n2+n1*n2+n0*n3+w0_psi*d1^2+ w0^2*d1*d2)/dm^2 + ...
(w1_psi*d1+w1*d2+w0_psi*d2+w0*d3)/dm);

```

```

% bound on derivative of w'''(x) with respect to psi
w3_psi = 8*prodq^4*(...
2*n2*n2/dm^2+2*n1*n3/dm^2 + ... %
4*n1*n2*d1/dm^3+2*n1^2*d2/dm^3+ ...%
n2^2/dm^2+ n1*n3/dm^2 + ... %
n1*n3/dm^2+n0*n4/dm^2 + ... %
2*n1*n2*d1/dm^3 + 2*n0*n3*d1/dm^3 + 2*n0*n2*d2/dm^3 +... %
4*n1*n2*d1/dm^3 + 2*n1^2*d2/dm^3 + ... %
2*n1*n2*d1/dm^3 + 2*n0*n3*d1/dm^3 + 2*n0*n2*d2/dm^3 + ... %
2*n1^2*d2/dm^3 + 2*n0*n2*d2/dm^3+2*n0*n1*d3/dm^3+ ... %
6*n1^2*d1^2/dm^4 + 6*n0*n2*d1^2/dm^4+6*n0*n1^2*d1*d2/dm^4 + ...
%
w2_psi*d1/dm + w2*d2/dm +... %
w1_psi*d2/dm + w1*d3/dm+ ... %
w1_psi*d1^2/dm^2 + w1^2*d1*d2/dm^2 + ... %
w1_psi*d2/dm + w1*d3/dm + ... %
w0_psi*d3/dm + w0*d4/dm + ... %
w0_psi*d1*d2/dm^2 + w0*d2*d2/dm^2 + w0*d1*d3/dm^2 + ... %
w1_psi*d1^2/dm^2 + w1^2*d1*d2/dm^2 + ... %
2*w0_psi*d1*d2/dm^2 + 2*w0*d2*d2/dm^2 + 2*w0*d1*d3/dm^2 + ...%
2*w0_psi*d1^3/dm^3 + 2*w0^3*d1^2*d2/dm^3 ... %
);

```

```

xi2 = xi*xi;
xi3 = xi2*xi;

% derivatives of v with respect to x
v1 = w1+ xi*w0;
v2 = w2+2*xi*w1+xi2*w0;
v3 = w3+3*xi*w2+3*xi2*w1+xi3*w0;

% derivatives with respect to psi
v1_psi = w1_psi+xi_der*w0+xi*w0_psi;
v2_psi = w2_psi+2*xi_der*w1+2*xi*w1_psi+2*xi*xi_der*w0+xi2*w0_psi;
v3_psi = w3_psi+3*xi_der*w2+3*xi*w2_psi+6*xi*xi_der*w1...
        +3*xi2*w1_psi + 3*xi2*xi_der*w0+xi3*w0_psi;

% bound on functions to interpolate
f1 = v1*v2;
f2 = v3*v2;
g = w0*v1;

% bound on derivative with respect to psi of
% functions to interpolate
f1_psi = v1_psi*v2+v1*v2_psi;
f2_psi = v3_psi*v2+v3*v2_psi;
g_psi = w0_psi*v1+w0*v1_psi;

% here = f1(round(0.5*length(f1)))

% bound on all sub bounds
out = nm(max(sup([f1,f2,g,f1_psi,f2_psi,g_psi])));

```

2.2 bound_numer_unstable.m

```
function M_x = bound_numer_unstable(dm,rho_x,a_q,b_q)
```

```

%
% bound for interpolation in x —————
%

```

```

pie = nm('pi');
% top of ellipse  $E_{\rho_\beta}$ 
max_abs_q = sup(real(b_q));
prodq = abs(log(a_q)/2);
max_imag_x = (rho_x-1/rho_x)/2;
max_real_psi = 0;

max_xi = (2*pie).*((1-a_q)./(1+a_q))+4*pie*(1./(1-b_q.^2)).*(b_q./(1-b_q));

max_xi_der = -4*pie*log(a_q)*(1/(1-b_q^2))*(b_q/(1-b_q)^2);

M_x = local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);

%-----
% local_bounds
%-----

function out = local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,xi,xi_der,prodq)

pie = nm('pi');
prod = pie/2;

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
n0 = temp(1);
n1 = temp(2); % first derivative
n2 = temp(3); % second derivative
n3 = temp(4); % third derivative
n4 = temp(5);

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
max_real_psi = 0;
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
d1 = temp(2); % first derivative
d2 = temp(3); % second derivative
d3 = temp(4); % third derivative

```

```
d4 = temp(5);
```

```
% bound on w(x), the conjugate of w(x), and their derivatives
% on the ellipse  $E_{\rho_x}$ 
```

```
% w(x)
w0 = n0^2/dm^2;
```

```
% w'(x)
w1 = 2*prod*( n0*n1/dm^2 + w0*d1/dm );
```

```
% w''(x)
w2 = 4*prod^2*(n1^2/dm^2+n0*n2/dm^2+2*n0*n1*d1/dm^3+w1*d1/dm+w0*d2/dm+w0*d1^2/dm^2);
```

```
% w'''(x)
w3 = 8*prod^3*( 2*n1*n2/dm^2+2*n1^2*d1/dm^3+n1*n2/dm^2+n0*n3/dm^2+2*n0*n2*d1/dm^3+...
  2*n1^2*d1/dm^3+2*n0*n2*d1/dm^3+2*n0*n1*d2/dm^3+6*n0*n1*d1^2/dm^4+w2*d1/dm-
+ w1*d2/dm+...
  w1*d1^2/dm^2+w1*d2/dm+w0*d3/dm+w0*d1*d2/dm^2+w1*d1^2/dm^2+2*w0*d1*d2/dm^2+...
  2*w0*d1^3/dm^3);
```

```
% bound on derivative of w(x) with respect to psi
w0_psi = 2*prodq*(n0*n1/dm^2);
```

```
% derivative of w'(x) with respect to psi
w1_psi = 2*prodq^2*(n0*n2/dm^2+n1^2/dm^2+w0_psi*d1/dm);
```

```
% bound on derivative of w''(x) with respect to psi
% —w''(x)— < 4*prod^2*( (2*n0*n1*d1)/dm^3+ (n1^2+n0*n2+w0*d1^2)/dm^2-
+ (w1*d1+ w0*d2)/dm );
w2_psi = 4*prodq^3*( (2*n1^2*d1+2*n0*n2*d1+2*n0*n1*d2)/dm^3 + ...
  (2*n1*n2+n1*n2+n0*n3+w0_psi*d1^2+ w0^2*d1*d2)/dm^2 + ...
  (w1_psi*d1+w1*d2+w0_psi*d2+w0*d3)/dm);
```

```
% bound on derivative of w'''(x) with respect to psi
w3_psi = 8*prodq^4*(...
  2*n2*n2/dm^2+2*n1*n3/dm^2 + ... %
```

```

4*n1*n2*d1/dm^3+2*n1^2*d2/dm^3+ ...%
n2^2/dm^2+ n1*n3/dm^2 + ... %
n1*n3/dm^2+n0*n4/dm^2 + ... %
2*n1*n2*d1/dm^3 + 2*n0*n3*d1/dm^3 + 2*n0*n2*d2/dm^3 +... %
4*n1*n2*d1/dm^3 + 2*n1^2*d2/dm^3 + ... %
2*n1*n2*d1/dm^3 + 2*n0*n3*d1/dm^3 + 2*n0*n2*d2/dm^3 + ... %
2*n1^2*d2/dm^3 + 2*n0*n2*d2/dm^3+2*n0*n1*d3/dm^3+ ... %
6*n1^2*d1^2/dm^4 + 6*n0*n2*d1^2/dm^4+6*n0*n1^2*d1*d2/dm^4 + ...
%
w2_psi*d1/dm + w2*d2/dm +... %
w1_psi*d2/dm + w1*d3/dm+ ... %
w1_psi*d1^2/dm^2 + w1^2*d1*d2/dm^2 + ... %
w1_psi*d2/dm + w1*d3/dm + ... %
w0_psi*d3/dm + w0*d4/dm + ... %
w0_psi*d1*d2/dm^2 + w0*d2*d2/dm^2 + w0*d1*d3/dm^2 + ... %
w1_psi*d1^2/dm^2 + w1^2*d1*d2/dm^2 + ... %
2*w0_psi*d1*d2/dm^2 + 2*w0*d2*d2/dm^2 + 2*w0*d1*d3/dm^2 + ...%
2*w0_psi*d1^3/dm^3 + 2*w0^3*d1^2*d2/dm^3 ... %
);

xi2 = xi*xi;
xi3 = xi2*xi;

% derivatives of v with respect to x
v1 = w1+ xi*w0;
v2 = w2+2*xi*w1+xi2*w0;
v3 = w3+3*xi*w2+3*xi2*w1+xi3*w0;

% derivatives with respect to psi
v1_psi = w1_psi+xi_der*w0+xi*w0_psi;
v2_psi = w2_psi+2*xi_der*w1+2*xi*w1_psi+2*xi*xi_der*w0+xi2*w0_psi;
v3_psi = w3_psi+3*xi_der*w2+3*xi*w2_psi+6*xi*xi_der*w1...
+3*xi2*w1_psi + 3*xi2*xi_der*w0+xi3*w0_psi;

% bound on functions to interpoate
f1 = v1*v2;
f2 = v3*v2;
g = w0*v1;

% bound on derivative with respect to psi of
% functions to interpolate
f1_psi = v1_psi*v2+v1*v2_psi;
f2_psi = v3_psi*v2+v3*v2_psi;

```

```

g_psi = w0_psi*v1+w0*v1_psi;

% bound on all sub bounds
out = nm(max(sup([f1,f2,g,f1_psi,f2_psi,g_psi])));

```

2.3 bound_sub_integrals.m

```

function [M_psi,M_x,M_q] = bound_sub_integrals(dm,rho_psi,rho_x,rho_q,a_q,b_q)

% constants
pie = nm('pi');
prod = pie/2;

% -----
% bound for interpolation in psi
% -----

% top of ellipse  $E_{\rho_x}$ 
max_imag_x = 0;
max_real_psi = (1+(rho_psi+1/rho_psi)/2)/2;
max_abs_q = sup(real(b_q));

vec = bound(prod,dm,max_imag_x,max_real_psi,max_abs_q);

% largest bound on all the sub integrands when  $\alpha = i$  beta
M_psi = sup(2*nm(max(vec)));

% -----
% bound for interpolation in x
% -----

% top of ellipse  $E_{\rho_x}$ 
max_imag_x = (rho_x-1/rho_x)/2;
max_real_psi = 1;
max_abs_q = sup(real(b_q));

```

```

vec = bound(prod,dm,max_imag_x,max_real_psi,max_abs_q);

% largest bound on all the sub integrands when alpha = i beta
M_x = max(vec);

% -----
% bound for interpolation in q
% -----

% top of ellipse  $E_{\rho_x}$ 
max_imag_x = 0;
max_real_psi = 1;
max_abs_q = (a_q+b_q)/2 + ((b_q-a_q)/4)*(rho_q+1/rho_q);

vec = bound(prod,dm,max_imag_x,max_real_psi,max_abs_q);

% largest bound on all the sub integrands when alpha = i beta
M_q = sup(2*nm(max(vec)));

% -----
% function for bounds
% -----
function vec = bound(prod,dm,max_imag_x,max_real_psi,max_abs_q)

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);

n0 = temp(1);
n1 = temp(2); % first derivative
n2 = temp(3); % second derivative
n3 = temp(4); % third derivative

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega' + n\omega + i\beta))$ 
max_real_psi = 0;
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
d1 = temp(2); % first derivative

```

```

d2 = temp(3); % second derivative
d3 = temp(4); % third derivative

```

```

% w(x)
w0 = n0^2/dm^2;

```

```

% w'(x)
w1 = 2*prod*( n0*n1/dm^2 + w0*d1/dm );

```

```

% w''(x)
w2 = 4*prod^2*(n1^2/dm^2+n0*n2/dm^2+2*n0*n1*d1/dm^3+w1*d1/dm+w0*d2/dm+w0*d1^2/dm^2);

```

```

% w'''(x)
w3 = 8*prod^3*( 2*n1*n2/dm^2+2*n1^2*d1/dm^3+n1*n2/dm^2+n0*n3/dm^2+2*n0*n2*d1/dm^3+...
  2*n1^2*d1/dm^3+2*n0*n2*d1/dm^3+2*n0*n1*d2/dm^3+6*n0*n1*d1^2/dm^4+w2*d1/dm-
+ w1*d2/dm+...
  w1*d1^2/dm^2+w1*d2/dm+ w0*d3/dm+w0*d1*d2/dm^2+w1*d1^2/dm^2+2*w0*d1*d2/dm^2+...
  2*w0*d1^3/dm^3);

```

```

% bounds on the 10 sub integrands

```

```

vec = [ w1*w2; ...
        w0*w2+2*w1*w1; ...
        w1*w0+2*w0*w1; ...
        w0*w0; ...
        w3*w2; ...
        3*w2*w2+2*w3*w1;
        w3*w0+6*w2*w1+3*w1*w2;
        3*w2*w0+6*w1*w1+w0*w2;
        3*w1*w0+2*w0*w1;
        w0*w0];

```

```

% bound on 10 sub integrands
vec = sup(vec);

```


2.4 bound_theta1_m.m

`function out = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,m)`

The first Jacobi Theta function is given by the series,

$$\vartheta_1(z) = 2 \sum_{n=0}^{\infty} (-1)^n q^{(n+1/2)^2} \sin((2n+1)z),$$

and its m th derivative is given by,

$$\vartheta_1^{(m)}(z) = 2 \sum_{n=0}^{\infty} (-1)^n q^{(n+1/2)^2} (2n+1)^m f((2n+1)z),$$

where $f(\cdot) = \sin(\cdot)$ if $m \equiv 0 \pmod{4}$, $f(\cdot) = \cos(\cdot)$ if $m \equiv 1 \pmod{4}$, $f(\cdot) = -\sin(\cdot)$ if $m \equiv 2 \pmod{4}$, and $f(\cdot) = -\cos(\cdot)$ if $m \equiv 3 \pmod{4}$.

Now for $m \geq 0$,

$$\begin{aligned} \left| (-1)^n q^{(n+1/2)^2} (2n+1)^m f((2n+1)z) \right| &\leq \left| q^{(n+1/2)^2} (2n+1)^m e^{(2n+1)|\Im(z)|} \right| \\ &\leq \left| q^{N^2+1/2} e^{(2N+1)|\Im(z)|} (2N+1)^m \right| q^n, \end{aligned}$$

for $n \geq N$ with N sufficiently large that

$$\left| q^{n^2+1/4} e^{(2n+1)|\Im(z)|} (2n+1)^m \right| \leq \left| q^{N^2+1/4} e^{(2N+1)|\Im(z)|} (2N+1)^m \right|,$$

whenever $n \geq N$. To determine how large N must be, we define,

$$g(x) := q^{x^2+1/4} e^{(2x+1)|\Im(z)|} (2x+1)^m.$$

We will take N large enough that $g'(x) < 0$ whenever $x \geq N$. If $m = 0$, then

$$g'(x) = 2q^{x^2+1/4} e^{(2x+1)|\Im(z)|} (x \log(q) + |\Im(z)|),$$

and we see that

$$N > -\frac{|\Im(z)|}{\log(q)}$$

suffices. If $M > 0$,

$$g'(x) = 2q^{x^2+1/4} e^{(2x+1)|\Im(z)|} ((x \log(q) + |\Im(z)|)(2x+1) + m).$$

From

$$(x \log(q) + |\Im(z)|)(2x + 1) + m < 0,$$

we find that

$$N > -\frac{2|\Im(z)| + \log(q) + \sqrt{(2|\Im(z)| + \log(q))^2 - 8\log(q)(|\Im(z)| + m)}}{4\log(q)}$$

suffices. For such an N , the error of the summation truncation is

$$q^{N^2+1/4} e^{(2N+1)|\Im(z)|} (2N+1)^m \frac{q^N}{1-q}.$$

```

%% constants
pie = nm('pi');
q = max_abs_q;
% Find max of  $\vartheta_1(\frac{\pi}{2\omega}(\omega x \pm i\omega' + n\omega + i\beta))$  on the ellipse  $E_\rho$ .

% abz = pie*(omega*max_imag_x+omega_prime+max_real_beta)/(2*omega);
abz = sup(pie*max_imag_x + abs(log(max_abs_q))*(1+max_real_psi))/2;

qlog = log(q);
one_fourth = nm(1)/4;
one_half = nm(1)/2;
con = q^one_fourth*exp(abz);

%%

% find N large enough that
%  $f(x) := q^{x^2+1/4} e^{(2x+1)|\Im(z)|} (2x+1)^m$ 
% is decreasing for  $x \geq N$ .
if m == 0
    N = ceil(sup(-abz/qlog));
else
    c1 = 2*abz+qlog;
    disc = c1^2-8*qlog*(abz+m);
    c2 = -c1/(4*qlog);
    c3 = disc/(4*qlog);

```

```

    N = ceil(sup(c2-c3));
end

% compute theta(z,q) and its derivatives with partial sum
%  $\vartheta_1^{(m)}(z) = 2 \sum_{n=0}^{N-1} (-1)^n q^{(n+1/2)^2} (2n+1)^m f((2n+1)z)$ 
out = nm(zeros(m+1,1));
for n = 0:N-1
    prod = 2*n+1;
    zprod = prod*abz;
    ezprod = exp(zprod);
    gen = ezprod*q^((n+one_half)^2);
    for ind = 1:m+1
        out(ind) = out(ind) + gen*prod^(ind-1);
    end
end

% truncation error
%  $q^{N^2+1/4} e^{(2N+1)|\Im(z)|} (2N+1)^m \frac{q^N}{1-q}$ 
% note that con =  $q^{1/4} e^{|\Im(z)|}$ 
for ind = 0:m
    out(ind+1) = out(ind+1) + con*q^(N^2)*exp(2*abz*N)*(2*N+1)^ind*q^N/(1-
q);
end

out = 2*out;
out = sup(out);

```

2.5 bound_xi.m

```

function out = bound_xi(a_psi,b_psi,rho_psi,a_q,b_q,rho_q,ntilde)

% latex description

```

Now

$$\begin{aligned}\xi(\alpha) &= 2i \left(\zeta(\alpha) - \frac{\alpha}{\omega} \zeta(\omega) \right) \\ &= 2i \left(\frac{\pi}{2\omega} \cot \left(\frac{\pi\alpha}{2\omega} \right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1-q^{2k}} \sin \left(\frac{k\pi\alpha}{\omega} \right) \right).\end{aligned}$$

When $\alpha = \omega + i\psi\omega'$ we have

$$\begin{aligned}\xi(\omega + i\psi\omega') &= 2i \left(\frac{\pi}{2\omega} \cot \left(\frac{\pi(\omega + i\psi\omega')}{2\omega} \right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1-q^{2k}} \sin \left(\frac{k\pi(\omega + i\psi\omega')}{\omega} \right) \right) \\ &= 2i \left(-\frac{\pi}{2\omega} \tan \left(\frac{i\pi\psi\omega'}{2\omega} \right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1-q^{2k}} (-1)^k \sin \left(\frac{ik\pi\psi\omega'}{\omega} \right) \right).\end{aligned}\tag{2.1}$$

When $\alpha = i\psi\omega'$ we have,

$$\xi(i\psi\omega') = 2i \left(\frac{\pi}{2\omega} \cot \left(\frac{\pi(i\psi\omega')}{2\omega} \right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1-q^{2k}} \sin \left(\frac{k\pi(i\psi\omega')}{\omega} \right) \right).$$

Note that if either $\psi \in [0, 1]$ and $0 < |q| < 1$ with $q \in \mathbb{C}$, or if $q \in [q_a, q_b] \subset (0, 1)$, $|\Re(\psi)| < 2$ and $|\Im(\psi)| < \frac{\pi}{|\log(q_a)|}$, then (3.2) is analytic in that region. If either $\psi \in (0, 1]$ and $0 < |q| < 1$ with $q \in \mathbb{C}$, or if $q \in [q_a, q_b] \subset (0, 1)$, $|\Re(\psi)| < 2$ and $|\psi| > 0$, then (3.3) is analytic.

Note that $\sin \left(\frac{i\pi\psi\omega'k}{\omega} \right) = \frac{1}{2i} (q^{\psi k} - q^{-\psi k})$ so that in both cases the infinite sum is bounded by

$$\begin{aligned}2 \sum_{k=1}^{\infty} \frac{q_0^{\gamma k}}{1-q_0^{2k}} &\leq \frac{2}{1-q_0^2} \sum_{k=1}^{\infty} q_0^{\gamma k} \\ &\leq \frac{2q_0^{\gamma}}{(1-q_0^2)(1-q_0^{\gamma})}.\end{aligned}$$

where $|q| \leq q_0 < 1$ and $\gamma := 2 - |\Re(\psi)|$.

Let M be a bound on $\tan \left(\frac{-i\psi \log(q)}{2} \right)$ or $\cot \left(\frac{-i\psi \log(q)}{2} \right)$, depending on the choice of α . Then

$$|\omega \xi(\tilde{n} + i\psi\omega')| \leq M\pi + \frac{8\pi q_0^{\gamma}}{(1-q_0^2)(1-q_0^{\gamma})}.$$

```

%%

pie = nm('pi');

%
% get bound on tan or cot
%

theta = nm(linspace(0,sup(2*pie)));
if rho_psi == 1
    psivec = nm(linspace(inf(a_psi),sup(b_psi)));
else
    psivec = nm(1)/2+ (rho_psi*exp(1i*theta)+exp(-1i*theta)/rho_psi)/4;
end

if rho_q == 1
    qvec = nm(linspace(inf(a_q),sup(b_q)));
else
    qvec = (a_q+b_q)/2+(b_q-a_q)*(rho_q*exp(1i*theta)+exp(-1i*theta)/rho_q)/4;
end

qv = nm(qvec(1:end-1),qvec(2:end));
psiv = nm(psivec(1:end-1),psivec(2:end));

if ntilde == 1
    % get bound on tan(1i*pi*psi*omega_prime/(2*omega))
    temp = sup(abs(tan(-1i*psiv.*log(qv)/2)));
elseif ntilde == 0
    % get bound on cot(1i*pi*psi*omega_prime/(2*omega))
    temp = sup(abs(cot(-1i*psiv.*log(qv)/2)));
end

if sum(sum(isnan(temp))) > 0
    error('NaN present');
end

```

```

M = max(max(temp));

max_abs_real_psi = (a_psi + b_psi)/2+(b_psi-a_psi)*(rho_psi +1/rho_psi)/4;
max_abs_q = (a_q + b_q)/2+(b_q-a_q)*(rho_q +1/rho_q)/4;

gamma = 2-max_abs_real_psi;
qg = max_abs_q^gamma;
out = pie*M+ (8*pie*qg)/((1-max_abs_q^2)*(1-qg));

```

2.6 bound_xi_der.m

```
function out = bound_xi_der(abs_q,min_abs_q,a,b,max_real_psi,rho_q,rho_psi,a_psi,b_psi,ntilde)
```

```
% latex comments
```

If $\tilde{n} = 1$, then from the q -series representation of the Weierstrass elliptic func-

tion,

$$\begin{aligned}
\frac{\partial}{\partial \psi} \xi(\tilde{n}\omega + i\psi\omega') &= 2\omega' \left(\wp(\tilde{\omega} + i\psi\omega') + \frac{\zeta(\omega)}{\omega} \right) \\
&= 2\omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \sec^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} \cos \left(\frac{ik\pi\psi\omega'}{\omega} \right) \right) \\
&= 2\omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \sec^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} \left(\frac{q^{\psi k} + q^{-\psi k}}{2} \right) \right).
\end{aligned} \tag{2.2}$$

If $\tilde{n} = 0$, then from the q -series representation of the Weierstrass elliptic function,

$$\begin{aligned}
\frac{\partial}{\partial \psi} \xi(\tilde{n}\omega + i\psi\omega') &= 2\omega' \left(\wp(\tilde{\omega} + i\psi\omega') + \frac{\zeta(\omega)}{\omega} \right) \\
&= 2\omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \csc^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} \frac{kq^{2k}}{1-q^{2k}} \cos \left(\frac{ik\pi\psi\omega'}{\omega} \right) \right) \\
&= 2\omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \csc^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} \frac{kq^{2k}}{1-q^{2k}} \left(\frac{q^{\psi k} - q^{-\psi k}}{2i} \right) \right).
\end{aligned} \tag{2.3}$$

Note that if either $\psi \in [0, 1]$ is fixed and $|q| < 1$ with $q \in \mathbb{C}$, or if $q \in [q_a, q_b] \subset (0, 1)$ is fixed and $|\Im(\psi)| < \frac{\pi}{|\log(q_a)|}$, then (3.2) is analytic in that region. Note that if either $\psi \in [0, 1]$ is fixed and $|q| < 1$ with $q \in \mathbb{C}$, or if $q \in [q_a, q_b] \subset (0, 1)$ is fixed and $\Re(\psi) > 0$, then (3.2) is analytic in that region.

Let $0 \leq q_0 < 1$, $\gamma \in \mathbb{R}$, and define

$$\begin{aligned}
f(x) &:= \sum_{k=0}^{\infty} \frac{1}{\gamma \log(q_0)} q_0^{\gamma k x} \\
&= \frac{1}{\gamma \log(q_0)} \frac{1}{1 - q_0^{\gamma x}}.
\end{aligned} \tag{2.4}$$

Note that

$$\begin{aligned}
f'(x) &= \sum_{k=0}^{\infty} k q_0^{\gamma k x} \\
&= \frac{q_0^{\gamma x}}{(1 - q_0^{\gamma x})^2}.
\end{aligned} \tag{2.5}$$

Then

$$\begin{aligned}
\left| \sum_{k=1}^{\infty} (-1)^{\tilde{n}k} \frac{kq^{2k}}{1-q^{2k}} (q^{\psi k} + q^{-\psi k}) \right| &\leq 2 \sum_{k=0}^{\infty} \frac{kq_0^{\gamma k}}{1-q_0^{2k}} \\
&\leq \frac{2}{1-q_0^2} \sum_{k=0}^{\infty} kq_0^{\gamma k} \\
&\leq \frac{2}{1-q_0^2} \frac{q_0^{\gamma}}{(1-q_0^{\gamma})^2},
\end{aligned} \tag{2.6}$$

where $|q| \leq q_0 < 1$ and $\gamma := 2 - \Re(\psi)$.

```
%%
```

```
%
% constants
%
```

```
pie = nm('pi');
gamma = 2-max_real_psi;
qg = abs_q^gamma;
```

```
% error check
if inf(gamma) <= 0
    error('max_psi too big');
end
```

```
%
% get bound on (sec/csc)(1i*pi*psi*omega_prime/(2*omega))
%
```

```
pnts = 100;
% bound on stadium in variable psi
qvec = nm(linspace(inf(a),sup(b),pnts));
theta = nm(linspace(0,sup(2*pie),pnts));
psivec = (a_psi+b_psi)/2+ (b_psi-a_psi)*(rho_psi*exp(1i*theta)+exp(-1i*theta)/rho_psi)/4;
qv = nm(qvec(1:end-1),qvec(2:end));
psiv = nm(psivec(1:end-1),psivec(2:end));
```



```

if ntilde == 1
    temp = sup(abs(sec(-1i*psiv.*log(qv)/2)));
elseif ntilde == 0
    temp = sup(abs(csc(-1i*psiv.*log(qv)/2)));
end
if sum(sum(isnan(temp))) > 0
    error('NaN present');
end
max_temp1 = max(max(temp));

% bound on stadium in variable q
psivec = nm(linspace(a_psi,b_psi,pnts));
theta = nm(linspace(0,sup(2*pie),pnts));
qvec = (a+b)/2+(b-a)*(rho_q*exp(1i*theta)+exp(-1i*theta)/rho_q)/4;
qv = nm(qvec(1:end-1),qvec(2:end));
psiv = nm(psivec(1:end-1),psivec(2:end));

if ntilde == 1
    temp = sup(abs(sec(-1i*psiv.*log(qv)/2)));
elseif ntilde == 0
    temp = sup(abs(csc(-1i*psiv.*log(qv)/2)));
end
if sum(sum(isnan(temp))) > 0
    error('NaN present');
end
max_temp2 = max(max(temp));

% take maximum of two bounds
temp_bd = nm(max(max_temp1,max_temp2));

%
% bound on infinite sum part
%

out = (2/(1-abs_q^2))*(qg/(1-qg)^2);

%
% combine all parts
%
```

```
out = 2*abs(log(min_abs_q))*pie*(out + temp_bd^2/4);
```

2.7 bound_xi_der_n1.m

```
function out = bound_xi_der_n1(abs_q,min_abs_q,a,b,max_real_psi,rho_q,rho_psi)
% out = bound_xi_der_n1(psi,rho_beta,omega,omega_prime)
%
% Find an upper bound on the derivative of xi with respect to beta
% when alpha = omega + i beta

% latex comments
```

Now

$$\begin{aligned}\frac{\partial}{\partial \beta} \xi(\omega + i\beta) &= 2 \left(\wp(\omega + i\beta) + \frac{\zeta(\omega)}{\omega} \right) \\ &= 2 \left(\left(\frac{\pi}{2\omega} \right)^2 \sec^2 \left(\frac{i\pi\beta}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1 - q^{2k}} \cos \left(\frac{ik\pi\beta}{\omega} \right) \right).\end{aligned}$$

Then if $\beta = \psi\omega'$,

$$\begin{aligned}\omega \frac{\partial}{\partial \psi} \xi(\omega + i\psi\omega') &= -2 \log(q) \left(\frac{\pi}{4} \sec^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \pi \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} (q^{\psi k} + q^{-\psi k}) \right) \\ &= -2 \log(q) \left(\frac{\pi \sec^2(-i\psi \log(q)/2)}{4} - \pi \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} (q^{\psi k} + q^{-\psi k}) \right).\end{aligned}\tag{2.7}$$

Note that if either $\psi \in [0, 1]$ and $0 < |q| < 1$ with $q \in \mathbb{C}$ or if $q \in [q_a, q_b] \subset (0, 1)$ and $|\Im(\psi)| < \frac{\pi}{|\log(q_a)|}$, then (3.2) is analytic in that region.

Let $0 \leq q_0 < 1$, $\gamma \in \mathbb{R}$, and define

$$\begin{aligned}f(x) &:= \sum_{k=0}^{\infty} \frac{1}{\gamma \log(q_0)} q_0^{\gamma k x} \\ &= \frac{1}{\gamma \log(q_0)} \frac{1}{1 - q_0^{\gamma x}}.\end{aligned}\tag{2.8}$$

Note that

$$\begin{aligned}f'(x) &= \sum_{k=0}^{\infty} k q_0^{\gamma k x} \\ &= \frac{q_0^{\gamma x}}{(1 - q_0^{\gamma x})^2}.\end{aligned}\tag{2.9}$$

Then in the above regions,

$$\begin{aligned}\left| \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} (q^{\psi k} + q^{-\psi k}) \right| &\leq 2 \sum_{k=0}^{\infty} \frac{kq_0^{\gamma k}}{1-q_0^{2k}} \\ &\leq \frac{2}{1-q_0^2} \sum_{k=0}^{\infty} k q_0^{\gamma k} \\ &\leq \frac{2}{1-q_0^2} \frac{q_0^{\gamma}}{(1-q_0^{\gamma})^2},\end{aligned}\tag{2.10}$$

where $|q| \leq q_0 < 1$ and $\gamma := 2 - \Re(\psi)$.

%%

```

%
% constants
%

pie = nm('pi');
gamma = 2-max_real_psi;
qg = abs_q^gamma;

% error check
if inf(gamma) <= 0
    error('max_psi too big');
end

%
% get bound on sec(1i*pi*psi*omega_prime/(2*omega))
%

qvec = nm(linspace(inf(a),sup(b)));
theta = nm(linspace(0,sup(2*pie)));
psivec = nm(1)/2+ (rho_psi*exp(1i*theta)+exp(-1i*theta)/rho_psi)/4;
qv = nm(qvec(1:end-1),qvec(2:end));
psiv = nm(psivec(1:end-1),psivec(2:end));
temp = sup(abs(sec(-1i*psiv.*log(qv)/2)));
if sum(sum(isnan(temp))) > 0
    error('NaN present');
end
max_sec1 = max(max(temp));

psivec = nm(linspace(0,1));
theta = nm(linspace(0,sup(2*pie)));
qvec = (a+b)/2+(b-a)*(rho_q*exp(1i*theta)+exp(-1i*theta)/rho_q)/4;
qv = nm(qvec(1:end-1),qvec(2:end));
psiv = nm(psivec(1:end-1),psivec(2:end));
temp = sup(abs(tan(-1i*psiv.*log(qv)/2)));
if sum(sum(isnan(temp))) > 0
    error('NaN present');
end
max_sec2 = max(max(temp));

sec_bd = nm(max(max_sec1,max_sec2));

```

```

%
% bound on infinite sum part
%

out = (2/(1-abs_q^2))*(qg/(1-qg)^2);

%
% combine all parts
%

out = 2*abs(log(min_abs_q))*pie*(out + sec_bd^2/4);

```

2.8 bound_xi_n1.m

```
function out = bound_xi_n1(q,rho_psi,max_tan)
```

```
% latex description
```

Now

$$\begin{aligned}\xi(\omega + i\beta) &= \frac{\pi}{2\omega} \cot\left(\frac{\pi(\omega + i\beta)}{2\omega}\right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1 - q^{2k}} \sin\left(\frac{k\pi(\omega + i\beta)}{\omega}\right) \\ &= -\frac{\pi}{2\omega} \tan\left(\frac{i\pi\beta}{2\omega}\right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1 - q^{2k}} (-1)^k \sin\left(\frac{ik\pi\beta}{\omega}\right).\end{aligned}$$

Note that

$$\begin{aligned}|\tan(z)| &= \left| \frac{e^{2iz} - 1}{i(e^{2iz} + 1)} \right| \\ &\leq \frac{1 + |e^{2iz}|}{|1 + e^{2iz}|} \\ &\leq \frac{1 + e^{-2\Im(z)}}{|1 + e^{2iz}|}.\end{aligned}$$

Let $x, y \in \mathbb{R}$ such that $2iz = x + iy$, and suppose that $|y| \leq \psi\pi$ where $0 \leq \psi < 1$. Now

$$\begin{aligned}|1 + e^{x+iy}|^2 &= 1 + 2e^x \cos(y) + e^{2x} \\ &\geq 1 + 2e^x \cos(\psi\pi) + e^{2x}.\end{aligned}$$

By simple calculus we find that $1 + 2e^x \cos(\psi\pi) + e^{2x}$ is bounded below by 1 if $\psi \leq \frac{1}{2}$, and is bounded below by $\sin(\psi\pi)$ if $\frac{1}{2} < \psi < 1$.

Then

$$|\tan(z)| \leq \frac{1 + e^{-2\Im(z)}}{\sin(\psi\pi)}$$

if $\frac{1}{2} < \psi < 1$, and

$$|\tan(z)| \leq 1 + e^{-2\Im(z)}$$

if $0 \leq \psi \leq \frac{1}{2}$.

Note that

$$\begin{aligned}\left| \sum_{k=1}^{\infty} \frac{q^{2k}}{1 - q^{2k}} (-1)^k \sin(kz) \right| &\leq \sum_{k=1}^{\infty} \frac{q^{2k}}{1 - q^{2k}} e^{k|\Im(z)|} \\ &= \sum_{k=1}^{\infty} \frac{\left(e^{|\Im(z)| - 2\pi\omega'/\omega} \right)^k}{1 - q^{2k}} \\ &\leq \frac{e^{|\Im(z)| - 2\pi\omega'/\omega}}{(1 - q^2)(1 - e^{|\Im(z)| - 2\pi\omega'/\omega})},\end{aligned}$$

so long as $|\Im(z)| - 2\pi\omega'/\omega < 0$.

Then if $\rho_\beta < 3 + 2\sqrt{2}$, we have

$$|\xi(\omega + i\beta)| \leq \frac{\pi(1 + e^V)}{2\omega \sin(\psi\pi)} + \frac{2\pi e^Q}{\omega(1 - q^2)(1 - e^Q)},$$

where

$$Q := \frac{\pi\omega'}{2\omega} \left(1 + \frac{1}{2} \left(\rho + \frac{1}{\rho} \right) \right) - \frac{2\pi\omega'}{\omega},$$

and

$$V = -2 \left(\frac{\pi\omega'}{4\omega} \right) \left(1 - \frac{1}{2} \left(\rho + \frac{1}{\rho} \right) \right).$$

```
pie = nm('pi');

% c1 = pie*omega_prime/omega;
c1 = -log(q);
c2 = (rho_psi + 1/rho_psi)/2;

Q = (c1/2)*(1+c2)-2*c1;

out = (pie/2)*max_tan + (2*pie)*exp(Q)/((1-q^2)*(1-exp(Q)));
out = nm(sup(out));
```

2.9 cf_biv_cheby.m

```
function cf = cf_biv_cheby(m,n,num_funs,fun)
% function cf_padau(m,n,fun)
%
% Returns the coefficients for bivariate Chebyshev interpolation
```

Let $f(x, y) : [-1, 1] \times [-1, 1] \rightarrow \mathbb{C}$. Let $T_i(x)$, $T_j(y)$ be Chebyshev polynomials of the first kind. This function returns a matrix `cf` of coefficients to the bivariate interpolation polynomial, $p(x, y) = \sum_{i=0}^m \sum_{j=0}^n c_{i,j} T_i(x) T_j(y)$, where $c_{i,j}$ is the i th + 1, j th + 1 row and column of the matrix `cf`.

```
% constants
pie = nm('pi');
c_1 = pie/(2*(m+1));
c_2 = pie/(2*(n+1));

% theta for x and y
theta_xr = c_1*(2*(0:1:m)+1);
theta_ys = c_2*(2*(0:1:n)+1);

% x and y interpolation points
xr = cos(theta_xr);
ys = cos(theta_ys);

% evaluate function at grid points
f_xy = nm(zeros(m+1,n+1,num_funs));

fun2 = (x)(fun(x,ys));
parfor r = 1:m+1

    %
    % start intlab in matlab workers when in parallel
    %

    clc;
    curr_dir = cd;
    %% startup commands
    cd('..');
    cd('..');
    startup('intlab','','start matlabpool','off');
    cd(curr_dir);
```



```

%
% get function values
%

f_xy(r, :, :) = fun2(xr(r));

end

% get Chebyshev polynomials evaluated at points;
Tx = cos((0:1:m).'*theta_xr);
Ty = cos(theta_ys.'*(0:1:n));

cf = nm(zeros(m+1,n+1,num_funs));

for j = 1:num_funs
    cf(:, :, j) = (4/((m+1)*(n+1))) * Tx * f_xy(:, :, j) * Ty;
    cf(:, 1, j) = cf(:, 1, j) / 2;
    cf(1, :, j) = cf(1, :, j) / 2;
end

```

2.10 cf_eval.m

```

function out = cf_eval(cf,x,y)
% function out = cf_eval(cf,x,y)
%
% Evaluates the two dimensional Chebyshev polynomial at (x,y)

```

Let $f(x, y) : [-1, 1] \times [-1, 1] \rightarrow \mathbb{C}$. Let $T_i(x)$, $T_j(y)$ be Chebyshev polynomials of the first kind. This function evaluates $p(x, y) = \sum_{i=0}^m \sum_{j=0}^n c_{i,j} T_i(x) T_j(y)$, where $c_{i,j}$ is the i th + 1, j th + 1 row and column of the matrix cf.

```

m = size(cf,1);
b = nm(zeros(m,length(y)));
for i = 1:m

```

```

    % evaluate  $b_i := \sum_{j=0}^n c_{i,j} T_j(y)$ 
    b(i,:) = single_cf(cf(i,:),y);
end

% evaluate  $\sum_{i=0}^m b_i T_i(x)$ 
out = single_cf(b.',x);

%-----
% single_cf(cf,z)
%-----
function out = single_cf(cf,z)

% get theta
theta = acos(z);
% evalaute  $\sum_{j=0}^n c_j \cos(j\theta)$ 

out = cos(theta.'*(0:1:size(cf,2)-1))*cf.';

```

2.11 distinct.m

```

clc; close all; beep off;

curr_dir = cd;
cd('..');
cd('..');
startup('intl'ab','start matlabpool','off');
format long;
clc;
cd(curr_dir);

```

```
% Spectral stability of periodic wave trains of the Korteweg-de
% Vries/Kuramoto-Sivashinsky equation in the Korteweg-de Vries limit
```

```
% display type
% intvalinit('DisplayMidRad');
intvalinit('DisplayInfSup');
%%
```

```
pie = nm('pi');
k = nm('0.99');
```

```
K = elliptic_integral(k,1);
E = elliptic_integral(k,2);
k2 = k*k;
c1 = 1-k2;
b1 = k2*K/(E-K)
b2 = k2*c1*K/(c1*K-E)
b3 = c1*K/E
```

2.12 instability.m

```
clear all; close all; beep off; clc; curr_dir = cd;
%% startup commands
cd('..');
cd('..');
startup('intl',",",'start matlabpool','off');
format long;
```

```

clc;
cd(curr_dir);
% Spectral stability of periodic wave trains of the Korteweg-de
% Vries/Kuramoto-Sivashinsky equation in the Korteweg-de Vries limit

```

```

% display type
intvalinit('DisplayMidRad');
% intvalinit('DisplayInfSup');

```

```

pie = nm('pi');
total_time = tic;

```

```

%%

```

```

% [dm,rho_x,q_min,q_max] = lower_bound_unstable
dm = nm('0.036927931316603');
rho_x = midrad( 5.46027719725235, 0.000000000000001);
q_min = nm('1e-7');
q_max = nm('0.4');

```

Note that ρ_ψ must be chosen so that $\xi(\omega + i\psi\omega')$ does not have a pole. The poles of ξ are $z = 2m\omega + 2n\omega'$. Setting $2m\omega + 2n\omega' = \omega + i\psi\omega'$ with $\psi = 1/2 + \tilde{\psi}/2$, we find that $|\Im(\tilde{\psi})| < -\frac{\pi}{\log(q)}$ is necessary and sufficient to ensure analyticity.

```

%%

```

```

% find rho_psi
c_psi = nm('0.9')*pie/abs(log(q_max));
rho_psi = nm(inf(c_psi + sqrt(c_psi^2+1)));
d.rho_psi = rho_psi;

```

```

% make sure rho_psi is small enough that bounds
% on xi(omega + li*psi*omega') are valid
if sup(rho_psi) >= 3+sqrt(nm(2))
    rho_psi = 3+sqrt(nm(2));

```

```

end

if sup(rho_psi) <= inf(4*sqrt(nm(15)))
    error('problem');
end

%
% get bounds for alpha = omega + 1i*psi*omega'
%

time1 = tic;

M_x = bound_numer_unstable(dm,rho_x,q_min,q_max);

abs_tol = 1e-17;
[N_x,err_x] = N_nodes(rho_x,M_x,abs_tol);

%-----
% get derivatives when ntilde = 1
%-----

ntilde = 1;
psi = 1;
fun =(q)(integrand_numer(N_x,err_x,q,psi,ntilde));

% k = nm('0.942202');

k = nm('0.9','0.901');

kappa = kappa_of_k(k);
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k^2),1);
omega = pie/kappa;
% omega_prime = elipk2*pie/(elipk*kappa);
% X = 2*pie/kappa;
% w1 = omega;
% w2 = omega_prime*1i;

```

```

% eta_omega = weierstrass_eta1(w1,w2);
q = exp(-pie*elipk2/elipk)

vals = fun(q);

numer_psi = imag(vals(:,2)+vals(:,4)./omega.^2)
g_psi = imag(vals(:,6))

numer_cond = any(inf(real(numer_psi))<=0);
denom_cond = any(inf(real(g_psi))<=0);

if (numer_cond+denom_cond) == 0
    fprintf('instability shown\n');
else
    error('instability not shown\n');
end

```

2.13 integrand_numer.m

```

function out = integrand_numer(Nx,err,q,psi,ntilde)

xi = xi_q_psi(q,psi,ntilde);
xi_der = xi_der_q_psi(q,psi,ntilde);

```

```

% hold on
% plot(mid(psi),mid(xi),'-k');
% plot(mid(psi),mid(xi_der),'-b');
% return

% plot(mid(psi),mid(xi),'k','MarkerSize',18);

pie = nm('pi');
half = nm(1)/2;
theta = ((0:1:Nx)+half)*pie/(Nx+1);
x = cos(theta);

% alpha = 0
J = theta_vec(q,0,x,3,0);
J = repmat(J,[1 length(psi)]);

E0 = 1./J(:,1);
E1 = -J(:,2)./J(:,1).^2;
E2 = 2*J(:,2).^2./J(:,1).^3-J(:,3)./J(:,1).^2;
E3 = -6*J(:,2).^3./J(:,1).^4+6*J(:,2).*J(:,3)./J(:,1).^3-J(:,4)./J(:,1).^2;

B0 = E0.*E0;
B1 = 2*E0.*E1;
B2 = 2*(E0.*E2+E1.*E1);
B3 = 2*(E3.*E0+3*E1.*E2);

% alpha \neq 0
L = theta_vec(q,psi,x,4,ntilde);

A0 = L(:,1).*L(:,1);
A1 = 2*L(:,1).*L(:,2);
A2 = 2*(L(:,1).*L(:,3)+L(:,2).*L(:,2));
A3 = 2*(L(:,4).*L(:,1)+3*L(:,2).*L(:,3));
A4 = 2*(L(:,1).*L(:,5)+4*L(:,2).*L(:,4)+3*L(:,3).*L(:,3));

w0 = A0.*B0;

```

```

w1 = A0.*B1+A1.*B0;
w2 = A0.*B2+2*A1.*B1+A2.*B0;
w3 = A0.*B3+3*A1.*B2+3*A2.*B1+ A3.*B0;

```

```

con = log(q)/(1i*pie);
D0 = con*(A1.*B0);
D1 = con*(A1.*B1+A2.*B0);
D2 = con*(A1.*B2+2*A2.*B1+A3.*B0);
D3 = con*(A1.*B3+3*A2.*B2+3*A3.*B1+ A4.*B0);

```

```

xicon = 1i*repmat(xi,length(x),1);
xicon_der = 1i*repmat(xi_der,length(x),1);

```

```

xicon2 = xicon.*xicon;
xicon3 = xicon.*xicon2;

```

```

v1 = w1+xicon.*w0;
v2 = w2+2*xicon.*w1+xicon2.*w0;
v3 = w3+3*xicon.*w2+3*xicon2.*w1+xicon3.*w0;

```

```

v1_psi = D1 + xicon_der.*w0+xicon.*D0;

```

```

v2_psi = D2+2*xicon_der.*w1+2*xicon.*D1+...
2*xicon.*xicon_der.*w0+xicon2.*D0;

```

```

v3_psi = D3 + 3*xicon_der.*w2+3*xicon.*D2+6*xicon.*xicon_der.*w1...
+3*xicon2.*D1+3*xicon2.*xicon_der.*w0+xicon3.*D0;

```

```

f1 = v1.*conj(v2);
f1_psi = v1_psi.*conj(v2)+v1.*conj(v2_psi);

```

```

f2 = v3.*conj(v2);
f2_psi = v3_psi.*conj(v2)+v3.*conj(v2_psi);

```



```

g = w0.*conj(v1);
g_psi = D0.*conj(v1)+w0.*conj(v1_psi);

% Chebyshev polynomials evaluated at the points
Tx = cos(theta.*(0:2:Nx));

cf1 = 2*f1.*Tx/(Nx+1);
cf1(:,1) = cf1(:,1)/2;
cf2 = 2*f2.*Tx/(Nx+1);
cf2(:,1) = cf2(:,1)/2;
cg = 2*g.*Tx/(Nx+1);
cg(:,1) = cg(:,1)/2;

cf1_psi = 2*f1_psi.*Tx/(Nx+1);
cf1_psi(:,1) = cf1_psi(:,1)/2;
cf2_psi = 2*f2_psi.*Tx/(Nx+1);
cf2_psi(:,1) = cf2_psi(:,1)/2;
cg_psi = 2*g_psi.*Tx/(Nx+1);
cg_psi(:,1) = cg_psi(:,1)/2;

out = nm(zeros(length(psi),1,6));

out(:,1) = 2*cf1*(1./(1-(0:2:Nx).^2)).';
out(:,3) = 2*cf2*(1./(1-(0:2:Nx).^2)).';
out(:,5) = 2*cg*(1./(1-(0:2:Nx).^2)).';

out(:,2) = 2*cf1_psi*(1./(1-(0:2:Nx).^2)).';
out(:,4) = 2*cf2_psi*(1./(1-(0:2:Nx).^2)).';
out(:,6) = 2*cg_psi*(1./(1-(0:2:Nx).^2)).';

% add integration ( in x ) error
out = out + 2*(nm(-err,err)+1i*nm(-err,err));

```

2.14 kappa_of_k.m

```
function kappa = kappa_of_k(k)
% function kappa = kappa_of_k(k)
%
% Returns kappa(k)
```

From

$$\left(\frac{K(k)\mathcal{G}(k)}{\pi} \right)^2 = \frac{7}{20} \frac{2(k^4 - k^2 + 1)E(k) - (1 - k^2)(2 - k^2)K(k)}{(-2 + 3k^2 + 3k^4 - 2k^6)E(k) + (k^6 + k^4 - 4k^2 + 2)K(k)}$$

we may determine $\kappa = \mathcal{G}(k)$.

```
pie = nm('pi');
% elliptic integrals
K = elliptic_integral(k,1);
E = elliptic_integral(k,2);
% KmE = elliptic_integral(k,3)

% kappa(k)
k2 = k.*k;
k4 = k2.*k2;
k6 = k2.*k4;
c1 = 2*(k4-k2+1).*E-(1-k2).*(2-k2).*K;
c2 = (-2+3*k2+3*k4-2*k6).*E+(k6+k4-4*k2+2).*K;
kappa = pie*sqrt(7*c1./(20*c2))./K;
```

2.15 lambda_xi.m

```
function [f,fd,fdd,g,gd,gdd] = lambda_xi(q,omega,omega_prime,psi,ntilde)
%
% out = lambda_xi(q,omega,ntilde,psi_tilde)
%
% Returns in the first three components
% xi(ntilde*omega_prime+1i*psi*omega_prime)
% and its first two derviatives with respect to psi. Returns in the next three
% components 1i*c*lambda_0(ntilde*omega+1i*psi*omega_prime) where c is a
% real, nonzero constant.
```

Now

$$\wp'(z) = -\frac{\sigma(2z)}{\sigma^4(z)}$$

$$\sigma(z) = \frac{2\omega}{\pi} e^{\eta_1 z^2/2\omega} \vartheta_1(\pi z/2\omega)/\vartheta_1'(0)$$

$$\implies \wp'(z) = -\frac{(\pi\vartheta_1'(0))^3}{8\omega^3} \frac{\vartheta_1(\pi z/\omega)}{\vartheta_1^4(\pi z/2\omega)}$$

```
%%
```

```
% pi
pie = nm('pi');
```

```
% \vartheta_1'(0)
v0 = theta_vec_x(q,0,1);
```

```
% number of derivatives to take
m = 2;
```

```

%  $\vartheta_1(\pi z/\omega)$ 
con = pie/(omega);
z = con*(ntilde*omega+1i*psi*omega_prime);
vn = theta_vec_z(q,z,m);

%  $\vartheta_1(\pi z/(2\omega))$ 
con = pie/(2*omega);
z = con*(ntilde*omega+1i*psi*omega_prime);
vd = theta_vec_z(q,z,m);

%  $\wp'(z)$ 
gdd = -v0(:,2).^3*pie^3.*vn(:,1)./(8*omega.^3.*vd(:,1).^4);
%  $\frac{\partial^2}{\partial \psi^2} \omega \zeta(\tilde{n}\omega + i\psi\omega')$ 
gdd = 2*1i*omega_prime^2*omega*gdd;

%  $\xi(\tilde{n} + i\psi\omega')$ 
g = xi_q_psi(q,psi,ntilde);
g = g.';

%  $\frac{\partial}{\partial \psi} \xi(\tilde{n} + i\psi\omega')$ 
gd = xi_der_q_psi(q,psi,ntilde);
gd = gd.';

% auxiliary quantities

A = ((pie/omega)*vd(:,1).^4.*vn(:,2)-(2*pie/omega)*vn(:,1).*vd(:,1).^3.*vd(:,2));

Ad = ( (4*pie/omega)*vd(:,1).^3.*vd(:,2).*vn(:,2)*(pie/(2*omega)) ...
+ (pie/omega)*vd(:,1).^4.*vn(:,3)*(pie/omega) ...
- (2*pie/omega)*vn(:,2)*(pie/omega).*vd(:,1).^3.*vd(:,2) ...
-(2*pie/omega)*vn(:,1).*3.*vd(:,1).^2.*vd(:,2)*(pie/(2*omega)).*vd(:,2)-
...
-(2*pie/omega)*vn(:,1).*vd(:,1).^3.*vd(:,3)*(pie/(2*omega))));

B = vd(:,1).^8;

```

```
Bd = 8*vd(:,1).^7.*vd(:,2)*(pie/(2*omega));
```

```
%  $ic\lambda_0(\tilde{n}\omega + i\psi\omega')$ 
f = 1i*vn(:,1)./vd(:,1).^4;
```

```
%  $\frac{\partial}{\partial\psi}ic\lambda_0(\tilde{n}\omega + i\psi\omega')$ 
fd = A./B;
fd = 1i*fd*(1i*omega_prime);
```

```
%  $\frac{\partial^2}{\partial\psi^2}ic\lambda_0(\tilde{n}\omega + i\psi\omega')$ 
fdd = (B.*Ad-A.*Bd)./B.^2;
fdd = 1i*fdd*(1i*omega_prime)^2;
```

2.16 lower_bound.m

```
function [dm,rho_x] = lower_bound(q)
```

```
frac = nm('0.9'); % strictly between 0 and 1 (for choosing rho_x)
```

```
dm = lower_bound_local(q,frac);
```

```
pie = nm('pi');
rho_con = -frac*log(q)/pie;
rho_x = rho_con+sqrt(rho_con^2+1);
```

```
%-----
% lower_bound_local
```

```

function out = lower_bound_local(q,frac)

steps = 8000;
pie = nm('pi');
% parity and mirror symmetry of  $\vartheta_1$  only requires we go
% to pi/2 instead of 2 pi.
half = nm('0.5');
con = (half*pie/steps);

ind = 0:1:steps;
theta = nm(ind(1:end-1)*con,ind(2:end)*con);

psi = 0; ntilde = 0;
fun = (q,x)(theta_vec(q,psi,x,0,ntilde));

rho_x = get_rho(q,frac);

x = half*(rho_x*exp(1i*theta)+exp(-1i*theta)/rho_x);

temp = abs(fun(q,x));

theta(129)
temp(129)

out = min(min(temp));

% check if lower bound is not helpful
if out == 0
    error('lower bound is 0');
end

% check that out is not NaN
if (out < 10)
    error('error apparent')
end

```

```

%-----
% get rho
%-----
function rho_x = get_rho(q,frac)

%
%% compute rho for ellipse used in getting Chebyshev bounds for x variable
%
```

To interpolate in $x \in [-1, 1]$ we need a bound on the integrands. The integrands are analytic in and on an ellipse that does not intersect zeros of $\vartheta_1(\pi(x \pm i\omega')/2)$. Now the zeros of ϑ_1 are the set $\{m\pi + n\pi i\omega'/\omega | m, n \in \mathbb{N}\}$. Then from

$$\frac{\pi}{2\omega}(\omega x \pm i\omega') = m\pi + n\pi i\omega'/\omega \quad (2.11)$$

we find

$$\Im(x) < \omega'/\omega. \quad (2.12)$$

is required.

Now if $0 < c < \omega'/\omega$ is the height of the top of the ellipse E_ρ , then

$$\frac{1}{2}(\rho - 1/\rho) = c, \quad (2.13)$$

then

$$\rho = c + \sqrt{c^2 + 1}. \quad (2.14)$$

```

pie = nm('pi');
rho_con = -frac*log(q)/pie;
rho_x = rho_con+sqrt(rho_con^2+1);
```

2.17 lower_bound_unstable.m

```
function [dm,rho_x,q_min_out,q_max_out] = lower_bound_unstable
```

```
temp = zeros(5,1);  
frac = nm('0.9'); % strictly between 0 and 1 (for choosing rho_x)
```

```
q_min = nm('0.1');  
q_max = nm('0.15');  
q_max_out = q_max;  
num_steps = 100;  
temp(1) = lower_bound_local(q_min,q_max,num_steps,frac)
```

```
q_min = nm('0.01');  
q_max = nm('0.1');  
num_steps = 100;  
temp(2) = lower_bound_local(q_min,q_max,num_steps,frac)
```

```
q_min = nm('0.001');  
q_max = nm('0.01');  
num_steps = 100;  
temp(3) = lower_bound_local(q_min,q_max,num_steps,frac)
```

```
q_min = nm('0.0001');  
q_max = nm('0.001');  
num_steps = 100;  
temp(4) = lower_bound_local(q_min,q_max,num_steps,frac)
```

```
q_min = nm('1e-7');
```



```

q_max = nm('0.0001');
q_min_out = q_min;
num_steps = 100;
temp(5) = lower_bound_local(q_min,q_max,num_steps,frac)

dm = min(temp);

pie = nm('pi');
rho_con = -frac*log(q_max)/pie;
rho_x = rho_con+sqrt(rho_con^2+1);

%-----
% lower_bound_local
%-----
function out = lower_bound_local(q_min,q_max,num_steps,frac)

steps = 8000;
pie = nm('pi');
% parity and mirror symmetry of  $\vartheta_1$  only requires we go
% to pi/2 instead of 2 pi.
half = nm('0.5');
con = (half*pie/steps);

ind = 0:1:steps;
theta = nm(ind(1:end-1)*con,ind(2:end)*con);

q_del = (q_max-q_min)/num_steps;
psi = 0; ntilde = 0;
fun = (q,x)(theta_vec(q,psi,x,0,ntilde));

out = 1000;
for j = 1:num_steps-1

    q = q_min + nm((j-1)*q_del,j*q_del);
    rho_x = get_rho(q,frac);

    x = half*(rho_x*exp(1i*theta)+exp(-1i*theta)/rho_x);

```

```

    out = min(out, min(inf(abs(fun(q,x)))));

end

% check if lower bound is not helpful
if out == 0
    error('lower bound is 0');
end

% check that out is not NaN
if (out < 10)
    error('error apparent')
end

%-----
% get rho
%-----
function rho_x = get_rho(q,frac)

%
%% compute rho for ellipse used in getting Chebyshev bounds for x variable
%
```

To interpolate in $x \in [-1, 1]$ we need a bound on the integrands. The integrands are analytic in and on an ellipse that does not intersect zeros of $\vartheta_1(\pi(x \pm i\omega')/2)$. Now the zeros of ϑ_1 are the set $\{m\pi + n\pi i\omega'/\omega | m, n \in \mathbb{N}\}$. Then from

$$\frac{\pi}{2\omega}(\omega x \pm i\omega') = m\pi + n\pi i\omega'/\omega \quad (2.15)$$

we find

$$\Im(x) < \omega'/\omega. \quad (2.16)$$

is required.

Now if $0 < c < \omega'/\omega$ is the height of the top of the ellipse E_ρ , then

$$\frac{1}{2}(\rho - 1/\rho) = c, \quad (2.17)$$

then

$$\rho = c + \sqrt{c^2 + 1}. \quad (2.18)$$

```
pie = nm('pi');
rho_con = -frac*log(q)/pie;
rho_x = rho_con+sqrt(rho_con^2+1);
```

2.18 numer.m

```
function out = numer(Nx,err,q,psi,ntilde)
% out = numer(Nx,err,q,psi,ntilde)
%
%
```

Definition of λ_1

$$\lambda_1 = \frac{\int_{-1}^1 \left[\frac{\partial}{\partial x} v(\omega x) + \frac{1}{\omega^2} \frac{\partial^3}{\partial x^3} v(\omega x) \right] \frac{\partial^2}{\partial x^2} \bar{v}(\omega x) dx}{\omega^2 \int_{-1}^1 v(\omega x) \frac{\partial}{\partial x} \bar{v}(\omega x) dx}. \quad (2.19)$$

Now

$$\begin{aligned} v(x) &= w(x)e^{\gamma x} \\ v'(x) &= w'(x)e^{\gamma x} + \gamma w(x)e^{\gamma x} \\ v''(x) &= w''(x)e^{\gamma x} + 2\gamma w'(x)e^{\gamma x} + \gamma^2 w(x)e^{\gamma x} \\ v'''(x) &= w'''(x)e^{\gamma x} + 3\gamma w''(x)e^{\gamma x} + 3\gamma^2 w'(x)e^{\gamma x} + \gamma^3 w(x)e^{\gamma x}. \end{aligned}$$

Then

$$\begin{aligned}
v(\omega x)\bar{v}'(\omega x) &= w(\omega x)\bar{w}'(\omega x) + (i\xi)w(\omega x)\bar{w}(\omega x), \\
v(\omega x)\bar{v}''(\omega x) &= \sum_{n=0}^3 \gamma^n c_n(x), \\
v'''(\omega x)\bar{v}''(\omega x) &= \sum_{n=0}^5 \gamma^n h_n(x),
\end{aligned} \tag{2.20}$$

where

$$\begin{aligned}
c_0(x) &:= w'(\omega x)\bar{w}''(\omega x) \\
c_1(x) &:= w(\omega x)\bar{w}''(\omega x) - 2w'(\omega x)\bar{w}'(\omega x) \\
c_2(x) &:= w'(\omega x)\bar{w}(\omega x) - 2w(\omega x)\bar{w}'(\omega x) \\
c_3(x) &:= w(\omega x)\bar{w}(\omega x) \\
h_0(x) &:= w'''(\omega x)\bar{w}''(\omega x) \\
h_1(x) &:= 3w''(\omega x)\bar{w}''(\omega x) - 2w'''(\omega x)\bar{w}'(\omega x) \\
h_2(x) &:= w'''(\omega x)\bar{w}(\omega x) - 6w''(\omega x)\bar{w}'(\omega x) + 3w'(\omega x)\bar{w}''(\omega x) \\
h_3(x) &:= 3w''(\omega x)\bar{w}(\omega x) - 6w'(\omega x)\bar{w}'(\omega x) + w(\omega x)\bar{w}''(\omega x) \\
h_4(x) &:= 3w'(\omega x)\bar{w}(\omega x) - 2w(\omega x)\bar{w}'(\omega x) \\
h_5(x) &:= w(\omega x)\bar{w}(\omega x).
\end{aligned} \tag{2.21}$$

%%

% _____
% Get theta and derivatives
% _____

% constants
pie = nm('pi');
half = nm(1)/2;

% interpolation points in x for integration
theta = ((0:1:Nx)+half)*pie/(Nx+1);
x = cos(theta);

% denominator term (alpha = 0)

```

J = theta_vec(q,0,x,3,0);
J = repmat(J,[1 length(psi)]);

E0 = 1./J(:,1);
E1 = -J(:,2)./J(:,1).^2;
E2 = 2*J(:,2).^2./J(:,1).^3-J(:,3)./J(:,1).^2;
E3 = -6*J(:,2).^3./J(:,1).^4+6*J(:,2).*J(:,3)./J(:,1).^3-J(:,4)./J(:,1).^2;

B0 = E0.*E0;
B1 = 2*E0.*E1;
B2 = 2*(E0.*E2+E1.*E1);
B3 = 2*(E3.*E0+3*E1.*E2);

% numerator term (alpha = ntilde*omega + 1i*psi*omega')
L = theta_vec(q,psi,x,4,ntilde);

A0 = L(:,1).*L(:,1);
A1 = 2*L(:,1).*L(:,2);
A2 = 2*(L(:,1).*L(:,3)+L(:,2).*L(:,2));
A3 = 2*(L(:,4).*L(:,1)+3*L(:,2).*L(:,3));

w0 = A0.*B0;
w1 = A0.*B1+A1.*B0;
w2 = A0.*B2+2*A1.*B1+A2.*B0;
w3 = A0.*B3+3*A1.*B2+3*A2.*B1+A3.*B0;

% -----
% functions in expansions
% -----

%  $v(\omega x)\bar{v}''(\omega x) = \sum_{n=0}^3 \gamma^n c_n(x)$ 
c0 = w1.*conj(w2);
c1 = w0.*conj(w2)-2*w1.*conj(w1);
c2 = w1.*conj(w0)-2*w0.*conj(w1);
c3 = w0.*conj(w0);

%  $v'''(\omega x)\bar{v}''(\omega x) = \sum_{n=0}^5 \gamma^n h_n(x)$ 
h0 = w3.*conj(w2);
h1 = 3*w2.*conj(w2)-2*w3.*conj(w1);
h2 = w3.*conj(w0)-6*w2.*conj(w1)+3*w1.*conj(w2);

```

```

h3 = 3*w2.*conj(w0)-6*w1.*conj(w1)+w0.*conj(w2);
h4 = 3*w1.*conj(w0)-2*w0.*conj(w1);
h5 = w0.*conj(w0);

% Chebyshev polynomials evaluated at the interpolation points
Tx = cos(theta.*(0:2:Nx));

% Chebyshev coefficients
cf_c0 = 2*c0.*Tx/(Nx+1);
cf_c0(:,1) = cf_c0(:,1)/2;

cf_c1 = 2*c1.*Tx/(Nx+1);
cf_c1(:,1) = cf_c1(:,1)/2;

cf_c2 = 2*c2.*Tx/(Nx+1);
cf_c2(:,1) = cf_c2(:,1)/2;

cf_c3 = 2*c3.*Tx/(Nx+1);
cf_c3(:,1) = cf_c3(:,1)/2;

cf_h0 = 2*h0.*Tx/(Nx+1);
cf_h0(:,1) = cf_h0(:,1)/2;

cf_h1 = 2*h1.*Tx/(Nx+1);
cf_h1(:,1) = cf_h1(:,1)/2;

cf_h2 = 2*h2.*Tx/(Nx+1);
cf_h2(:,1) = cf_h2(:,1)/2;

cf_h3 = 2*h3.*Tx/(Nx+1);
cf_h3(:,1) = cf_h3(:,1)/2;

cf_h4 = 2*h4.*Tx/(Nx+1);
cf_h4(:,1) = cf_h4(:,1)/2;

```

```

cf_h5 = 2*h5.*Tx/(Nx+1);
cf_h5(:,1) = cf_h5(:,1)/2;

out = nm(zeros(length(psi),1,10));
out(:,1) = 2*cf_c0*(1./(1-(0:2:Nx).^2)).';
out(:,2) = 2*cf_c1*(1./(1-(0:2:Nx).^2)).';
out(:,3) = 2*cf_c2*(1./(1-(0:2:Nx).^2)).';
out(:,4) = 2*cf_c3*(1./(1-(0:2:Nx).^2)).';
out(:,5) = 2*cf_h0*(1./(1-(0:2:Nx).^2)).';
out(:,6) = 2*cf_h1*(1./(1-(0:2:Nx).^2)).';
out(:,7) = 2*cf_h2*(1./(1-(0:2:Nx).^2)).';
out(:,8) = 2*cf_h3*(1./(1-(0:2:Nx).^2)).';
out(:,9) = 2*cf_h4*(1./(1-(0:2:Nx).^2)).';
out(:,10) = 2*cf_h5*(1./(1-(0:2:Nx).^2)).';

% add integration ( in x) error
out = out + 2*(nm(-err,err)+1i*nm(-err,err));

```

2.19 simplicity.m

```

clear all; close all; beep off; clc; curr_dir = cd;
%% startup commands
cd('..');
cd('..');
startup('intlab','start matlabpool','off');
format long;
clc;
cd(curr_dir);

% display type
intvalinit('DisplayMidRad');
% intvalinit('DisplayInfSup');

%
%% parameters
%

```

```

k = nm('0.99');

%
%% derived constants
%

p.k = k;
pie = nm('pi');
kappa = kappa_of_k(k);
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k^2),1);
omega = pie/kappa;
omega_prime = elipk2*pie/(elipk*kappa);
X = 2*pie/kappa;
w1 = omega;
w2 = omega_prime*1i;
eta_omega = weierstrass_eta1(w1,w2);
q = exp(-pie*elipk2/elipk);

%
% —————
%

ntilde = 1;

num = 100;
L = 50;   %  $\beta_0 = 0.98$ 
L2 = 1;   %  $\beta \in [0, \omega']$ 

psi_x = nm(0:1:num)/(L2*num)+nm(L2-1)/L2;
psi_x = nm(psi_x(1:end-1),psi_x(2:end)); % make intervals
psi_y = nm(0:1:num)/(L*num)+nm(L-1)/L;
psi_y = nm(psi_y(1:end-1),psi_y(2:end)); % make intervals
num = num -1;

% get components of h(x,y)
[f1,fd1,fd1,g1,gd1,gdd1] = lambda_xi(q,omega,omega_prime,psi_x,ntilde);

```



```

F1 = repmat(f1,1,num+1);
Fd1 = repmat(fd1,1,num+1);
Fdd1 = repmat(fdd1,1,num+1);
G1 = repmat(g1,1,num+1);
Gd1 = repmat(gd1,1,num+1);
Gdd1 = repmat(gdd1,1,num+1);

%
% -----
%

ntilde = 0;

% get components of h(x,y)
[f2,fd2,fdd2,g2,gd2,gdd2] = lambda_xi(q,omega,omega_prime,psi_y,ntilde);
f2 = f2.'; fd2 = fd2.'; fdd2 = fdd2.'; g2 = g2.'; gd2 = gd2.'; gdd2 = gdd2.';

F2 = repmat(f2,num+1,1);
Fd2 = repmat(fd2,num+1,1);
Fdd2 = repmat(fdd2,num+1,1);
G2 = repmat(g2,num+1,1);
Gd2 = repmat(gd2,num+1,1);
Gdd2 = repmat(gdd2,num+1,1);

%
% -----
%

% Show that  $\frac{\partial}{\partial \psi} \xi(\omega + i\psi\omega') > 0$  for  $\psi \in [0, 1]$ .
min_gd1 = min(inf(real(gd1)));
fprintf('\n\nThe minimum of xi(omega + 1i*beta) for beta in [0,omega"] is:
%16.16g\n\n',min_gd1);

con = 1;

```

```

T = con*(F1+F2);
Tx = con*(Fd1);
Txx = con*(Fdd1);
% Txy = 0
Ty = con*(Fd2);
Tyy = con*(Fdd2);

S = G1+G2-2*pie;
Sx = Gd1;
% Sxy = 0
Sxx = Gdd1;
Sy = Gd2;
Syy = Gdd2;

h = T.^2+S.^2;
hx = 2*T.*Tx+2*S.*Sx;
hxx = 2*Tx.^2+2*T.*Txx+2*Sx.^2+2*S.*Sxx;
hy = 2*T.*Ty+2*S.*Sy;
hyy = 2*Ty.^2+2*T.*Tyy+2*Sy.^2+2*S.*Syy;
hxy = 2*Ty.*Tx+2*Sy.*Sx;

%Del

Del = (hxx.*hyy-hxy.^2);
Del = inf(real(Del));

if abs(sum(sum(isfinite(Del)-1))) > 0
    error('Del is not finite');
end

min_Del = min(min(Del));
fprintf('\n\nThe minimum of hxx*hyy-hxy^2: %16.16g\n',min_Del);

% hxx

hxx = inf(real(hxx));

```

```

if abs(sum(sum(isfinite(hxx)-1))) > 0
    error('hxx is not finite');
end

min_hxx = min(min(hxx));
fprintf('The minimum of hxx: %16.16g\n',min_hxx);

temp1 = -4*gdd1/(2*1i*omega_prime^2*omega);
max_gdd1 = max(sup(abs(imag(temp1))));

temp2 = -4*gdd2(1)/(2*1i*omega_prime^2*omega);
gdd1_one = inf(imag(temp2));

fprintf('Max abs(lambda_0(omega+1i*psi*omega_prime)): %16.16g\n',max_gdd1);
fprintf('Inf of lambda_0(1i*psi_0*omega_prime): %16.16g\n',gdd1_one);
fprintf('Diff: %16.16g\n',inf(nm(gdd1_one)-nm(max_gdd1)))
fprintf('Inf 3*omega*pi/X: %16.16g\n', inf(3*pie/2));
fprintf('Sup of xi(1i*psi_0*omega_prime: %16.16g\n', sup(real(g2(1))));
fprintf('Diff of last two: %16.16g\n\n', -inf(g2(1)-3*pie/2));

% throw errors if we do not verify the necessary properties

if min_Del <=0
    error('Del not positive');
end

if min_hxx <= 0
    error('hxx not positive');
end

if inf(nm(gdd1_one)-nm(max_gdd1)) <= 0
    error('condition fails')
end

if -inf(g2(1)-3*pie/2) <= 0

```

```

    error('condition fails')
end

```

2.20 stability.m

```

clear all; close all; beep off; clc; curr_dir = cd;
%% startup commands
cd('..');
cd('..');
startup('intl'ab','start matlabpool','off');
format long;
clc;
cd(curr_dir);
% Spectral stability of periodic wave trains of the Korteweg-de
% Vries/Kuramoto-Sivashinsky equation in the Korteweg-de Vries limit

% display type
% intv'init('DisplayMidRad');
intv'init('DisplayInfSup');

total_time = tic;

%%

k = nm('0.99');
p.k = k;
pie = nm('pi');
kappa = kappa_of_k(k);
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k^2),1);
omega = pie/kappa;
omega_prime = elipk2*pie/(elipk*kappa);
X = 2*pie/kappa;
w1 = omega;
w2 = omega_prime*1i;
eta_omega = weierstrass_eta1(w1,w2);
q = exp(-pie*elipk2/elipk);

```

```

pie = nm('pi');

%
% Get the lower bound of theta
%
```

```

t1 = tic;
[dm,rho_x] = lower_bound(q);
d.dm_time = toc(t1);
d.dm = dm;
d.rho_x = rho_x;
```

Note that ρ_ψ must be chosen so that $\xi(\omega + i\psi\omega')$ does not have a pole. The poles of ξ are $z = 2m\omega + 2n\omega'$. Setting $2m\omega + 2n\omega' = \omega + i\psi\omega'$ with $\psi = 1/2 + \tilde{\psi}/2$, we find that $|\Im(\tilde{\psi})| < -\frac{\pi}{\log(q)}$ is necessary and sufficient to ensure analyticity.

```
%%
```

```

% find rho_psi
c_psi = nm('0.9')*pie/abs(log(q));
rho_psi = nm(inf(c_psi + sqrt(c_psi^2+1)));
d.rho_psi = rho_psi;
```

```

% make sure rho_psi is small enough that bounds
% on xi(omega + 1i*psi*omega') are valid
if sup(rho_psi) >= 3+sqrt(nm(2))
    rho_psi = 3+sqrt(nm(2));
end
```

```

if sup(rho_psi) <= 1
    error('problem');
end
```

```

%
% find rho_q
%
```

```

q_min = q;
q_max = q+1e-12;
a = q_min;
b = q_max;
d.q_min = q_min;
d.q_max = q_max;

rho1 = (b+a)/(b-a)-2*sqrt(a*b)/(b-a);
rho2 = (b+a)/(b-a)+2*sqrt(a*b)/(b-a);
rho3 = (2-a-b)/(b-a)-2*sqrt(1-a-b+a*b)/(b-a);
rho4 = (2-a-b)/(b-a)+2*sqrt(1-a-b+a*b)/(b-a);

rho_left = nm(rho1,rho3);
rho_right = nm(rho2,rho4);
if sup(rho_left) >= inf(rho_right)
    error('problem');
end

rho_left = nm(sup(rho_left));
rho_right = nm(inf(rho_right));
rho_q = rho_left + nm('0.9')*(rho_right-rho_left);
d.rho_q = rho_q;

%
% get bounds for alpha = omega + 1i*psi*omega'
%

% ntilde = 1
ntilde = 1;

time1 = tic;

[M_psi,M_x,M_q] = bound_numer(dm,rho_psi,rho_x,rho_q,q_min,q_max,nm(0),nm(1),ntilde);
toc(time1)

abs_tol = 1e-17;
[N_x,err_x] = N_nodes(rho_x,M_x,abs_tol);

```

```

abs_tol = 1e-17;
[N_psi,err_psi] = N_nodes(rho_psi,M_psi,abs_tol);

[N_q,err_q] = N_nodes(rho_q,M_q,abs_tol);

d.bd_n1_time = toc(time1);

%-----
% get interpolation coefficients when ntilde = 1
%-----

a_q = a;
b_q = b;
a_psi = nm(0);
b_psi = nm(1);

fun = (q,psi)(integrand_numer(N_x,err_x,q*(b_q-a_q)/2+(a_q+b_q)/2,...
    psi*(b_psi-a_psi)/2+(a_psi+b_psi)/2,ntilde));

% interpolation coefficients
t1 = tic;
cfn1 = cf_biv_cheby(N_q,N_psi,6,fun);
d.cf1_time = toc(t1);
d.cfn1 = cfn1;

d.M_psi_n1 = M_psi;
d.M_q_n1 = M_q;
d.M_x_n1 = M_x;
d.N_psi_n1 = N_psi;
d.N_q_n1 = N_q;
d.N_x_n1 = N_x;
d.err_psi_n1 = err_psi;
d.err_q_n1 = err_q;

```

```

d.err_x_n1 = err_x;
d.a_q = a_q;
d.b_q = b_q;
d.cfn1 = cfn1;
d.dm = dm;
d.rho_psi_n1 = rho_psi;
d.rho_q_n1 = rho_q;
d.rho_x_n1 = rho_x;

%
% get bounds on 10 integrands for alpha = 1i*psi*omega'
%

[M_psi_10,M_x_10,M_q_10] = bound_sub_integrals(dm,rho_psi,rho_x,rho_q,a,b);

d.M_psi_10 = M_psi_10;
d.M_x_10 = M_x_10;
d.M_q_10 = M_q_10;

abs_tol = 1e-17;
[N_x_10,err_x_10] = N_nodes(rho_x,M_x_10,abs_tol);
d.N_x_10 = N_x_10;
d.err_x_10 = err_x_10;

abs_tol = 1e-4;
[N_psi_10,err_psi_10] = N_nodes(rho_psi,M_psi_10,abs_tol);
d.N_psi_10 = N_psi_10;
d.err_psi_10 = err_psi_10;

[N_q_10,err_q_10] = N_nodes(rho_q,M_q_10,abs_tol);
d.N_q_10 = N_q_10;
d.err_q_10 = err_q_10;

%-----
% get interpolation coefficients when ntilde = 0
% for 10 integrands
%-----

```



```

a_q = a;
b_q = b;
a_psi = nm(0);
b_psi = nm(1);

ntilde = 0;
fun = (q,psi)(numer(N_x,err_x,q*(b_q-a_q)/2+(a_q+b_q)/2,...
    psi*(b_psi-a_psi)/2+(a_psi+b_psi)/2,ntilde));

% interpolation coefficients
t1 = tic;
cf10 = cf_biv_cheby(N_q_10,N_psi_10,10,fun);
d.cf10_time = toc(t1);
d.cf10 = cf10;

%-----
% Find bound on numerator when ntilde = 0
%-----

ntilde = 0;

a_psi = inf(nm('0.5'));
b_psi = 1;

time1 = tic;
[M_psi,M_x,M_q] = bound_numer...
    (dm,rho_psi,rho_x,rho_q,q_min,q_max,a_psi,b_psi,ntilde);
toc(time1)

abs_tol = 1e-17;
[N_x,err_x] = N_nodes(rho_x,M_x,abs_tol);

abs_tol = 1e-17;
[N_psi,err_psi] = N_nodes(rho_psi,M_psi,abs_tol);

[N_q,err_q] = N_nodes(rho_q,M_q,abs_tol);

```

```

d.bd_n1_time = toc(time1);

d.M_psi_n0 = M_psi;
d.M_q_n0 = M_q;
d.M_x_n0 = M_x;
d.N_psi_n0 = N_psi;
d.N_q_n0 = N_q;
d.N_x_n0 = N_x;
d.err_psi_n0 = err_psi;
d.err_q_n0 = err_q;
d.err_x_n0 = err_x;

%-----
% get interpolation coefficients when ntilde = 0
%-----

fun = (q,psi)(integrand_numer(N_x,err_x,q*(b_q-a_q)/2+(a_q+b_q)/2,...
    psi*(b_psi-a_psi)/2+(a_psi+b_psi)/2,ntilde));

% interpolation coefficients
t1 = tic;
cfn0 = cf_biv_cheby(N_q,N_psi,6,fun);
d.cf0_time = toc(t1);
d.cfn0 = cfn0;

%-----
% save data
%-----

d.total_time = toc(total_time);

file_name = 'd_11Nov2013';
saveit(curr_dir,'interval_arithmetic',d,file_name,'data');

```

2.21 theta_vec.m

```
function out = theta_vec(q,psi,x,m,ntilde)
% out = theta_vec(q,psi,x,m,ntilde)
%
% Returns an interval enclosure of f(x) and its first m derivatives where
% f(x) = v(pie(x + n)/2+i*pi*omega_prime*(1+psi)/(2*omega))
% and v(z) is the first Jacobi Theta function with nome q.
%
% The input should satisfy -1 <= x <= 1, 0 <= psi <= 1, 0<q<1, ntilde = 0
or 1, m >= 0
```

The first Jacobi Theta function is given by the series,

$$\vartheta_1(z) = 2 \sum_{n=0}^{\infty} (-1)^n q^{(n+1/2)^2} \sin((2n+1)z).$$

From

$$\vartheta_1(z) = 2 \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} \sin((2n-1)z),$$

we find that

$$\begin{aligned} f(x) &:= \vartheta_1 \left(\frac{\pi}{2\omega} (\omega x + i\omega' + \tilde{n}\omega + i\psi\omega') \right) \\ &= -i \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n-1)} \right), \end{aligned}$$

where $\hat{v} := e^{i\pi(x+\tilde{n})/2} q^{(1+\psi)/2}$. Hence

$$\frac{\partial^m}{\partial x^m} f(x) = -i(i\pi/2)^m \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} (2n-1)^m \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n-1)} \right).$$

We find that

$$\begin{aligned}
Err &:= \left| -i(i\pi/2)^m \sum_{n=N+1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} (2n-1)^m \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n+1)} \right) \right| \\
&\leq 2q^{1/4}(\pi/2)^m \sum_{n=N+1}^{\infty} q^{n^2/2} \left((2n-1)^m q^{n^2/2-n-(2n-1)(1+\psi)/2} \right) \\
&\leq 2q^{1/4}(\pi/2)^m \sum_{n=N+1}^{\infty} q^{n^2/2} \\
&\leq 2q^{1/4}(\pi/2)^m q^{(N+1)^2/2} \frac{1}{1-q},
\end{aligned}$$

as long as we take N large enough so that $g(x) := \left((2x-1)^m q^{n^2/2-x-(2x-1)(1+\psi)/2} \right)$ satisfies $g(N) < 1$, $g(x) < 0$ whenever $x > N$.

%%

%% Error checking

%

% I tested against theta1_m for accuracy, which was tested against Maple and Mathematica. I

% found that they agreed. In testing, I found that this function has error radius about twice that of

% theta1_m for $m = 4$, but that it computed much faster. For example, it took nearly 2 minutes to

% evaluate theta1_m on 200 x points but only took about 0.2 seconds with this vectorized version.

%

% Below is code used for testing:

% x = nm(linspace(-1,1,30));

% q = nm('0.5');

% psi = nm(linspace(0,1,20));

% xicon = 0;

% xicon_der = 0;

%

% tic

% val = integrands(x,q,psi,xicon,xicon_der);

% toc

%

% pie = nm('pi');

% con = pie/2;

% diff = 0;

% for j = 1:length(x)

```

%   for k = 1:length(psi)
%
%       z = pie*(x(j) + 1)/2-log(q)*1i*(1+psi(k))/(2);
%       temp = theta1_m(z,q,4,1e-17);
%       for ind = 1:5
%           diff = max(diff,sup(temp(ind)*con^(ind-1)-val(j,k,ind)));
%       end
%   end
% end
%
% disp(diff);
% 5.975892748039020e-11 + 5.958839722380777e-11i

%%

% error target
tol = 1e-17;

%
% constants
%

psi0 = max(sup(psi));
pie = nm('pi');
one = nm(1);
req_1 = sup(real(-2*m/log(q)));
c1 = nm(2+psi0);
c2 = nm(2*q^(one/4)*(pie/2)^m/(1-q));
qsqrt = q^(one/2);
half = one/2;

%
% find N large enough that truncation error is less than tol
%

err = tol + 1;
N = 0;
maxit = 1000;
while err > tol
    N = N + 1;

```

```

    if N > maxit
        error('maximum iterations exceeded');
    end

    % check that derivative of  $f(x) := q^{x^2/2-x-(2x-1)(1+\psi)/2}(2x-1)^m$  is
    decreasing for  $x \geq N$ .
    if inf(real((N-c1)*(2*N-1))) <= req-1
        continue
    end

    % check that  $f(x) := q^{N^2/2-N-(2N-1)(1+\psi)/2}(2N-1)^m \leq 1$ 
    if sup(real(q^(N^2/2-N-(2*N-1)*(1+psi0)/2)*(2*N-1)^m)) > 1
        continue
    end

    % turncation error
    err = sup(real(c2*qsqrt^((N+1)^2)));

end

%
% evaluate the partial sum
%

% initialize vectors
out = nm(zeros(length(x),length(psi),m+1));
temp = out;

% rearrange dimensions if needed
sx = size(x,1);
if sx == 1
    x = x.';
end

sx = size(psi,1);
if sx > 1
    psi = psi.';
end

%
% add partial sum

```

```

%

vhat = exp(1i*pi*(x+ntilde)/2)*q.^((1+psi)/2);

for n = 1:N

    vtemp = vhat.^(2*n-1);
    for k = 0:m
        temp(:,k+1) = (2*n-1)^k*(vtemp - (-1)^k*vtemp.^(-1));
    end

    out = out + (-1)^(n+1)*q^((n-half)^2)*temp;

end

% complete differentiation rule
for j = 0:m
    out(:,j+1) = out(:,j)*(1i*pi/2)^j;
end

% multiply by -1i coming from sin definition and add error interval
out = -1i*out + nm(-err,err)+1i*nm(-err,err);

```

2.22 theta_vec_z.m

```

function out = theta_vec_z(q,z,m)

% error target
tol = 1e-17;

%
% constants
%

```

```

sig = -nm(max(sup(abs(imag(z)))))/nm(min(inf(abs(log(q)))));

q0 = nm(max(sup(q)));

%
% find N large enough that truncation error is less than tol
%

% Choose N so that derivative of  $f(x) := q_0^{x^2+1/4+(2x+1)\sigma}(2x+1)^m$  is decreasing
% for  $x \geq N$ .
c = m/log(q0);
temp = -(2*sig+1)+sqrt((2*sig+1)^2-8*(sig+c))/4;
N = ceil(sup(temp));
quarter = nm(1)/4;
half = nm(1)/2;

err = tol + 1;
maxit = 1000;
while err > tol
    N = N + 1;
    if N > maxit
        error('maximum iterations exceeded');
    end

    % turncation error
    err1 = q0^((N+1)^2+quarter+(2*(N+1)+1)*sig)*(2*(N+1)+1)^m;
    err2 = q0^(N+1)/(1-q0);
    err = sup(err1*err2);

end

%
% evaluate the partial sum
%

% initialize vectors

```



```

out = nm(zeros(length(z),length(q),m+1));

% rearrange dimensions if needed
sz = size(z,1);
if sz == 1
    z = z.';
end

sq = size(q,2);
if sq == 1
    q = q.';
end

%
% add partial sum
%

V = exp(1i*z);

temp = out;

for n = 0:N
    qtemp = (-1)^n*q.^((n+half)^2);

    for k = 0:m
        temp(:,k+1) = (2*n+1)^k*(V.^(2*n+1) - (-1)^k*V.^(-(2*n+1)))*qtemp;
    end

    out = out + temp;
end

% complete differentiation rule

```

```

for j = 0:m
    out(:,j+1) = out(:,j+1)*(1i)^(j-1);
end

% multiply by -1i coming from sin definition and add error interval
out = out + nm(-err,err)+1i*nm(-err,err);

```

2.23 verify_data_n0.m

```

function verify_data_n0

clear all; close all; beep off; clc; curr_dir = cd;
%% startup commands
cd('..');
cd('..');
startup('intlab','start matlabpool','off');
format long;
clc;
cd(curr_dir);
% Spectral stability of periodic wave trains of the Korteweg-de
% Vries/Kuramoto-Sivashinsky equation in the Korteweg-de Vries limit

% display type
% intvallinit('DisplayMidRad');
intvallinit('DisplayInfSup');

file_name = 'd_31Oct2013'; curr_dir = cd;

ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data');
d = ld.var;

k = nm('0.99');

```

```
j = 1; % just one case since single wave
```

```
left_point = 1e-3;  
left_verify(k,d,left_point,j)
```

```
a = left_point;  
b = 0.1;  
[left1,right1] = verify(k,d,a,b,j)
```

```
a = 0.09;  
b = 0.9;  
[left2,right2] = verify(k,d,a,b,j);
```

```
a = 0.89;  
b = 0.96;  
[left3,right3] = verify(k,d,a,b,j);
```

```
a = 0.95;  
b = 0.99;  
[left4,right4] = verify(k,d,a,b,j);
```

```
a = 0.989;  
b = 1;  
[left5,right5] = verify(k,d,a,b,j)
```

```
if inf(right1-left2) <=0  
    error('not verified');  
end
```

```
if inf(right2-left3) <=0  
    error('not verified');  
end
```

```
if inf(right3-left4) <=0  
    error('not verified');
```

```

end

if inf(right4-left5) <=0
    error('not verified');
end

%
% verify on right
%

%TODO: Find exact bounds
ntilde = 0;
Nx = 250
err = 1e-16
pie = nm('pi');
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
q = exp(-pie*elipk2./elipk);
psi_end = linspace(right5,1,1000);
psi_end = nm(psi_end(1:end-1),psi_end(2:end));
vals = integrand_numer(Nx,err,q,psi_end,ntilde);
kappa = kappa_of_k(k);
omega = pie./kappa;
numer_der = vals(:, :, 2)+vals(:, :, 4)/omega^2;

if any(sup(imag(numer_der))>=0) == 1
    error('not verified on right');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [left, right] = verify(k,d,a,b,j)

pie = nm('pi');

cf = d.cf10;

%
% plot
%
```

```

d.b_psi = 1;
d.a_psi = 0;

lam_10_q = (2/pie)*log(d.N_q_10)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_10));

err = d.err_q_10+lam_10_q*d.err_psi_10;

psi = linspace(a,b,1000);
psi = nm(psi);
psi = nm(psi(1:end-1),psi(2:end));

c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
c1_psi = 2/(d.b_psi-d.a_psi);
c2_psi = (d.a_psi+d.b_psi)/2;

kappa = kappa_of_k(k);
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
q = exp(-pie*elipk2./elipk);
omega = pie./kappa;
q_tilde = c1_q*(q-c2_q);
psi_tilde = c1_psi*(psi-c2_psi);

omega = omega.';
omega = repmat(omega,[1 length(psi)]);
%%%%%%%%%%

c0 = (cf_eval(cf(:,1),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
c1 = (cf_eval(cf(:,2),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
c2 = (cf_eval(cf(:,3),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
c3 = (cf_eval(cf(:,4),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
h0 = (cf_eval(cf(:,5),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
h1 = (cf_eval(cf(:,6),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
h2 = (cf_eval(cf(:,7),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
h3 = (cf_eval(cf(:,8),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
h4 = (cf_eval(cf(:,9),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);
h5 = (cf_eval(cf(:,10),q_tilde,psi_tilde))+nm(-err,err)+li*nm(-err,err);

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
xicon = 1i*xi_q_psi(q(j),psi,0);  
numer1 = c0(j,:) + c1(j,:).*xicon.^1+ c2(j,:).*xicon.^2 + c3(j,:).*xicon.^3;  
numer2 = h0(j,:) + h1(j,:).*xicon.^1+ h2(j,:).*xicon.^2 + h3(j,:).*xicon.^3 + ...  
        h4(j,:).*xicon.^4 + h5(j,:).*xicon.^5;
```

```
numer = numer1 + numer2./omega(j,:).^2;
```

```
numer = imag(numer);
```

```
if sum(isnan(numer))>0  
    error('NaN present');  
end
```

```
ind1 = find(inf(numer)>0);
```

```
if isempty(ind1)  
    error('nothing verified');  
end
```

```
bg = ind1(1);  
for jk = 0:length(ind1)-1  
    if norm(bg+jk-ind1(jk+1)) > 0  
        error('not consecutive');  
    end  
end
```

```
left = nm(sup(psi(ind1(1))));  
right = nm(inf(psi(ind1(end))));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

function out = left_verify(k,d,b,j)

out = 0;

pie = nm('pi');
cf = d.cf10;
d.b_psi = 1;
d.a_psi = 0;

lam_10_q = (2/pie)*log(d.N_q-10)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q-10));

err = d.err_q_10+lam_10_q*d.err_psi_10;

psi = linspace(0,b,1000);
psi = nm(psi);
psi = nm(psi(1:end-1),psi(2:end));

c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
c1_psi = 2/(d.b_psi-d.a_psi);
c2_psi = (d.a_psi+d.b_psi)/2;

kappa = kappa_of_k(k);
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
q = exp(-pie*elipk2./elipk);
omega = pie./kappa;
q_tilde = c1_q*(q-c2_q);
psi_tilde = c1_psi*(psi-c2_psi);

omega = omega.';
omega = repmat(omega,[1 length(psi)]);
%%%%%%%%%%%%

c0 = (cf_eval(cf(:,1),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);

```

```

c1 = (cf_eval(cf(:,2),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
c2 = (cf_eval(cf(:,3),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
c3 = (cf_eval(cf(:,4),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
h0 = (cf_eval(cf(:,5),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
h1 = (cf_eval(cf(:,6),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
h2 = (cf_eval(cf(:,7),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
h3 = (cf_eval(cf(:,8),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
h4 = (cf_eval(cf(:,9),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
h5 = (cf_eval(cf(:,10),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);

p0 = max(sup(abs(c0+h0./omega(j,:).^2)))
p1 = max(sup(abs(c1+h1./omega(j,:).^2)))
p2 = max(sup(abs(c2+h2./omega(j,:).^2)))
p3 = max(sup(abs(c3+h3./omega(j,:).^2)))
p4 = max(sup(abs(h4./omega(j,:).^2)))
p5 = min(inf(abs(h5./omega(j,:).^2)))

R = nm(1)+(1/nm(p5))*nm(max([p0,p1,p2,p3,p4]));
R = sup(R)
xicon = abs(xi_q_psi(q(j),b,0))

if inf(real(xicon)) > R
    out = 1;
end

```

2.24 verify_data_n1.m

```

clear all; close all; beep off; clc; curr_dir = cd;
%% startup commands
cd('..');
cd('..');
startup('intlabb', 'start matlabpool', 'off');
format long;
clc;
cd(curr_dir);
% Spectral stability of periodic wave trains of the Korteweg-de
% Vries/Kuramoto-Sivashinsky equation in the Korteweg-de Vries limit

```



```

% display type
intvalinit('DisplayMidRad');
% intvalinit('DisplayInfSup');
pie = nm('pi');

file_name = 'd_31Oct2013'; curr_dir = cd;

ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data');
d = ld.var;

cf = d.cfn1;

%-----
% plot
%-----
d.b_psi = 1;
d.a_psi = 0;

lam_n1_q = (2/pie)*log(d.N_q_n1)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n1));

err = d.err_q_n1+lam_n1_q*d.err_psi_n1;

k = nm('0.99');

pie = nm('pi');

num = 300;
psi_vals = linspace(0,1,10*num);
psi_vals = nm(psi_vals);
psi_vals = nm(psi_vals(1:end-1),psi_vals(2:end));

c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;

```

```

c1_psi = 2/(d.b_psi-d.a_psi);
c2_psi = (d.a_psi+d.b_psi)/2;

kappa = kappa_of_k(k);
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
q = exp(-pie*elipk2./elipk);
omega = pie./kappa;
q_tilde = c1_q*(q-c2_q);
psi_tilde = c1_psi*(psi_vals-c2_psi);

% get f1 interpolated value
f1 = imag(cf_eval(cf(:,1),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
% get f1_psi interpolated value
f1_psi = (cf_eval(cf(:,2),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
% get f2 interpolated value
f2 = imag(cf_eval(cf(:,3),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
% get f2_psi interpolated value
f2_psi = (cf_eval(cf(:,4),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
% get g interpolated value
g = imag(cf_eval(cf(:,5),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
% get g_psi interpolated value
g_psi = (cf_eval(cf(:,6),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);

g = real(g);
g_psi = imag(g_psi);

omega = omega.';
omega = repmat(omega,[1 length(psi_vals)]);

numer = real(f1+f2./omega.^2);
numer_psi = imag(f1_psi + f2_psi./omega.^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j = 1:length(q)

    numer_psi_left_min = min( inf(numer_psi(1:num)))
    numer_mid_min = min(inf(numer(j,num+1:9*num)))
    numer_psi_right_max = max(sup(numer_psi(9*num+1:10*num-1)))

```

```

    if any(inf(numer_psi(1:num))<= 0)
        error('failed to verify');
    end
    if any(inf(numer(j,num+1:9*num))<= 0)
        error('failed to verify');
    end
    if any(sup(numer_psi(9*num+1:10*num-1))>=0)
        error('failed to verify');
    end
end

for j = 1:length(q)

    g_psi_left_max = max((sup(g_psi(1:num))))
    g_mid_max = max(sup(g(j,num+1:9*num)))
    g_psi_right_min = min(inf(g_psi(9*num+1:10*num-1)))

    if any(sup(g_psi(1:num))>= 0)
        error('failed to verify');
    end
    if any(sup(g(j,num+1:9*num))>= 0)
        error('failed to verify');
    end
    if any(inf(g_psi(9*num+1:10*num-1))<=0)
        error('failed to verify');
    end
end
end

```

2.25 xi_der_q_psi.m

```

function out = xi_der_q_psi(q,psi,ntilde)
% function out = xi_der_q_psi(q,psi,ntilde)

```

%
 % returns the derivative of omega*xi with respect to psi

Now

$$\begin{aligned}\frac{\partial}{\partial \psi} \omega \xi(\omega + i\psi \omega') &= 2\omega \omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \sec^2 \left(\frac{i\pi \psi \omega'}{2\omega} \right) - \frac{\pi^2}{\omega^2} \sum_{k=1}^{\infty} (-1)^k \frac{k q^{2k}}{1 - q^{2k}} (q^{\psi k} + q^{-\psi k}) \right) \\ &= \frac{-\pi \log(q)}{2} \sec^2 \left(\frac{\psi \log(q) i}{2} \right) + 2\pi \log(q) \sum_{k=1}^{\infty} (-1)^k \frac{k q^{2k}}{1 - q^{2k}} (q^{\psi k} + q^{-\psi k})\end{aligned}$$

Note that

$$\begin{aligned}\left| \sum_{k=N+1}^{\infty} (-1)^k \frac{k q^{2k}}{1 - q^{2k}} (q^{\psi k} + q^{-\psi k}) \right| &\leq \sum_{k=N+1}^{\infty} \frac{k q^{2k}}{1 - q^{2k}} (q^{\psi k} + q^{-\psi k}) \\ &\leq \frac{2}{1 - q^{2(N+1)}} \sum_{k=0}^{\infty} (N+1+k) q^{(2-\psi)(N+1+k)} \\ &\leq \frac{2(N+1) q^{(2-\psi)(N+1)}}{1 - q^{2(N+1)}} \sum_{k=0}^{\infty} q^{(2-\psi)k} + \frac{2q^{(2-\psi)N}}{1 - q^{2(N+1)}} \sum_{k=0}^{\infty} k q^{(2-\psi)k} \\ &\leq \frac{2(N+1) q^{(2-\psi)(N+1)}}{1 - q^{2(N+1)}} \frac{1}{1 - q^{(2-\psi)}} + \frac{2q^{(2-\psi)N}}{1 - q^{2(N+1)}} \frac{q^{(2-\psi)}}{(1 - q^{(2-\psi)})^2}\end{aligned}$$

%%

%
 % constants
 %

```
pie = nm('pi');
psi0 = max(sup(psi));
q0 = abs(q^(2-psi0));
qs0 = q0^(2-sup(abs(psi0)));
q02 = q0*q0;
```

```

%
% find N for given error
%

N = 0;
maxit = 1000;
tol = 1e-17;
err = tol + 1;
while err > tol
    N = N+1;
    if N > maxit
        error('Maximum iterations exceeded');
    end
    temp = 2*(N+1)*qs0^(N+1)/((1-q02^(N+1))*(1-qs0)) + 2*qs0^(N+1)/((1-
q02^(N+1))*(1-qs0)^2);
    err = sup(temp);
end

% initialize output
out = nm(zeros(1,length(psi)));

% find partial sum
q2 = q*q;
for k = 1:N
    q2k = q2^k;
    out = out + ((-1)^ntilde)^k*(k*q2k/(1-q2k))*(q.^(k*psi)+q.^(-k*psi));
end

% add error
out = out + nm(-err,err)+1i*nm(-err,err);

% add non infinite sum parts
if ntilde == 0
    out = (-pie*log(q)/2)./(sin(1i*log(q)*psi/2).^2) + 2*pie*log(q)*out;
elseif ntilde == 1
    out = (-pie*log(q)/2)./(cos(1i*log(q)*psi/2).^2) + 2*pie*log(q)*out;
end

```

2.26 xi-q_psi.m

```
function out = xi_q_psi(q,psi,ntilde)
% computes omega*xi(ntilde*omega + 1i*psi*omega_prime)
```

From

$$\vartheta_1(z) = 2 \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} \sin((2n-1)z),$$

we find that

$$\begin{aligned} f(x) &:= \vartheta_1 \left(\frac{\pi}{2\omega} (\omega x + i\omega' + \tilde{n}\omega + i\psi\omega') \right) \\ &= -i \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n+1)} \right), \end{aligned}$$

where $\hat{v} := e^{i\pi(x+\tilde{n})/2} q^{(1+\psi)/2}$. Hence

$$\frac{\partial^m}{\partial x^m} f(x) = -i(i\pi/2)^m \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} (2n-1) \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n+1)} \right).$$

We find that

$$\begin{aligned} Err &:= \left| -i(i\pi/2)^m \sum_{n=N+1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} (2n-1) \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n+1)} \right) \right| \\ &\leq 2q^{1/4} (\pi/2)^m \sum_{n=N+1}^{\infty} q^{n^2/2} \left((2n-1)^m q^{n^2/2 - n - (2n-1)(1+\psi)/2} \right) \\ &\leq 2q^{1/4} (\pi/2)^m \sum_{n=N+1}^{\infty} q^{n^2/2} \\ &\leq 2q^{1/4} (\pi/2)^m q^{(N+1)^2/2} \frac{1}{1-q}, \end{aligned}$$

as long as we take N large enough so that $g(x) := \left((2x-1)^m q^{n^2/2 - x - (2x-1)(1+\psi)/2} \right)$ satisfies $g(N) < 1$, $g(x) < 0$ whenever $x > N$.

```
%%%
```

```

%
% constants
%

pie = nm('pi');
psi0 = max(sup(psi));
q0 = abs(q^(2-psi0));

%
% find N for given error
%

N = 0;
maxit = 1000;
tol = 1e-17;
err = tol + 1;
while err > tol
    N = N+1;
    if N > maxit
        error('Maximum iterations exceeded');
    end
    temp = q0^(N+1)/((1-q^(2*(N+1)))*(1-q0));
    err = sup(temp);
end

% initialize output
out = nm(zeros(1,length(psi)));

% add partial sum
q2 = q*q;
logq = log(q);
for k = 1:N
    q2k = q2^k;
    out = out + (q2k/(1-q2k))*sin(k*(pie*ntilde-li*psi*logq));
end

% add error
out = out + nm(-err,err) + li*nm(-err,err);

```



```
% multiply by constants
```

```
out = 2*pi*out;
```

```
% add non infinite sum parts
```

```
out = out + (pi/2)*cot(pi*ntilde/2-1i*psi*logq/2);
```

```
% multiply by constants to get omega*xi
```

```
out = 2*1i*out;
```

Chapter 3

Full study

3.1 N_nodes.m

```
function [N,err] = N_nodes(rho,M,abs_tol)
% N = N_nodes(rho,M,abs_tol)
```

Determine the number of Chebyshev nodes, N , needed to interpolate the analytic function $f(z)$ on the interval $z \in [-1, 1]$ with absolute interpolation error no more than abs_tol , where $|f(z)| \leq M$ for $z = \frac{1}{2}(\rho e^{i\theta} + e^{-i\theta}/\rho)$, with $\theta \in [0, 2\pi]$.

L_ρ is the length of the ellipse E_ρ on which the bound M is computed. A bound is given by $L_\rho \leq \pi \sqrt{\rho^2 + 1/\rho^2}$. The distance D_ρ between the ellipse E_ρ and the line $[-1, 1]$ is no smaller than $(\rho + 1/\rho)/2 - 1$. An interpolation bound is given by, $|f(x) - p(x)| \leq M L_\rho / (2\pi D_\rho \sinh(\eta) \sinh(\eta N))$ where $\eta = \log \rho$.

```
eta = log(rho);
denom = ((rho+1/rho)/2-1);
bound_constant = M*sqrt(rho^2+1/rho^2)/denom;
```

```
N = 3;
err = bound_constant/sinh(eta*(N+1));
while err > 0.1*abs_tol
    N = N+1;
    err = bound_constant/sinh(eta*(N+1));
end
```

3.2 agm.m

```
function cnew = agm(a,b)
% function out = agm(a,b)
%
% Evaluate the arithmetic geometric mean of a and b
% using interval arithmetic.
%
% Input should be non-negative: a >= 0 and b >= 0.

%
% check for user error
%

% check that input is real
if max(sup(abs(imag(a)))) > 0
    error(' a must be real');
end

if max(sup(abs(imag(b)))) > 0
    error(' b must be real');
end

% check that input is non-negative
if min(inf(a)) < 0
    error(' a >= 0 required');
end

if min(inf(b)) < 0
    error(' b >= 0 required');
end

% make initial interval containing AGM(a,b)
cold = hull(nm(a/2,b/2),nm(2*a,2*b));
cnew = hull(a,b);
```

```

% iterate until agm has converged
while (max(sup(cold))-min(inf(cold)))-(max(sup(cnew))-min(inf(cnew))) > 0

    cold = cnew;
    atemp = (a+b)/2;
    btemp = sqrt(a.*b);
    a = atemp;
    b = btemp;
    cnew = hull(a,b);

end

```

3.3 bound_numer.m

```

function [M_psi,M_x,M_q] = bound_numer(dm,rho_psi,rho_x,rho_q,a_q,b_q,a_psi,b_psi,ntilde)

%
% constants
%

min_abs_q = (a_q+b_q)/2 -(b_q-a_q)*(rho_q+1/rho_q)/4;

%
% bound for interpolation in psi
%

% top of ellipse  $E_{\rho_\psi}$ 
max_imag_x = 0;
max_real_psi = (1+(rho_psi+1/rho_psi)/2)/2;
max_abs_q = sup(real(b_q));

```

```

prodq = abs(log(a_q)/2);

max_xi = bound_xi(a_psi,b_psi,rho_psi,a_q,b_q,1,ntilde);
max_xi_der = bound_xi_der...
    (max_abs_q,min_abs_q,a_q,b_q,max_real_psi,rho_q,rho_psi,a_psi,b_psi,ntilde);

M_psi = 2*local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);

%
% bound for interpolation in x -----
%

% top of ellipse  $E_{\rho_\beta}$ 
max_imag_x = (rho_x-1/rho_x)/2;
max_real_psi = 1;
% max_abs_q the same
% prodq the same
% max_xi the same
% max_xi_der the same

M_x = local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);

%
% bound for interpolation in q -----
%

% top of ellipse  $E_{\rho_\beta}$ 
max_imag_x = 0;
max_real_psi = 1;
max_abs_q = (a_q+b_q)/2 + ((b_q-a_q)/4)*(rho_q+1/rho_q);

max_xi = bound_xi(a_psi,b_psi,rho_psi,a_q,b_q,1,ntilde);
max_xi_der = bound_xi_der...
    (max_abs_q,min_abs_q,a_q,b_q,max_real_psi,rho_q,rho_psi,a_psi,b_psi,ntilde);

min_abs_q = (a_q+b_q)/2 - (b_q-a_q)*(rho_q+1/rho_q)/4;
prodq = abs(log(min_abs_q)/2);

```

```
M_q = 2*local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);
```

```
%-----
% local_bounds
%-----
```

```
function out = local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,xi,xi_der,prodq)
```

```
pie = nm('pi');
prod = pie/2;
```

```
% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
n0 = temp(1);
n1 = temp(2); % first derivative
n2 = temp(3); % second derivative
n3 = temp(4); % third derivative
n4 = temp(5);
```

```
% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
max_real_psi = 0;
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
d1 = temp(2); % first derivative
d2 = temp(3); % second derivative
d3 = temp(4); % third derivative
d4 = temp(5);
```

```
% bound on  $w(x)$ , the conjugate of  $w(x)$ , and their derivatives
% on the ellipse  $E_{\rho_x}$ 
```

```
%  $w(x)$ 
w0 = n0^2/dm^2;
```

```
%  $w'(x)$ 
w1 = 2*prod*( n0*n1/dm^2 + w0*d1/dm );
```

```

% w''(x)
w2 = 4*prod^2*(n1^2/dm^2+n0*n2/dm^2+2*n0*n1*d1/dm^3+w1*d1/dm+w0*d2/dm+w0*d1^2/dm^2);

% w'''(x)
w3 = 8*prod^3*( 2*n1*n2/dm^2+2*n1^2*d1/dm^3+n1*n2/dm^2+n0*n3/dm^2+2*n0*n2*d1/dm^3+...
  2*n1^2*d1/dm^3+2*n0*n2*d1/dm^3+2*n0*n1*d2/dm^3+6*n0*n1*d1^2/dm^4+w2*d1/dm-
+ w1*d2/dm+...
  w1*d1^2/dm^2+w1*d2/dm+ w0*d3/dm+w0*d1*d2/dm^2+w1*d1^2/dm^2+2*w0*d1*d2/dm^2+...
  2*w0*d1^3/dm^3);

% bound on derivative of w(x) with respect to psi
w0_psi = 2*prodq*(n0*n1/dm^2);

% derivative of w'(x) with respect to psi
w1_psi = 2*prodq^2*(n0*n2/dm^2+n1^2/dm^2+w0_psi*d1/dm);

% bound on derivative of w''(x) with respect to psi
% —w''(x)— < 4*prod^2*( (2*n0*n1*d1)/dm^3+(n1^2+n0*n2+w0*d1^2)/dm^2-
+ (w1*d1+ w0*d2)/dm );
w2_psi = 4*prodq^3*( (2*n1^2*d1+2*n0*n2*d1+2*n0*n1*d2)/dm^3 + ...
  (2*n1*n2+n1*n2+n0*n3+w0_psi*d1^2+ w0^2*d1*d2)/dm^2 + ...
  (w1_psi*d1+w1*d2+w0_psi*d2+w0*d3)/dm);

% bound on derivative of w'''(x) with respect to psi
w3_psi = 8*prodq^4*(...
  2*n2*n2/dm^2+2*n1*n3/dm^2 + ... %
  4*n1*n2*d1/dm^3+2*n1^2*d2/dm^3+ ...%
  n2^2/dm^2+ n1*n3/dm^2 + ... %
  n1*n3/dm^2+n0*n4/dm^2 + ... %
  2*n1*n2*d1/dm^3 + 2*n0*n3*d1/dm^3 + 2*n0*n2*d2/dm^3 +... %
  4*n1*n2*d1/dm^3 + 2*n1^2*d2/dm^3 + ... %
  2*n1*n2*d1/dm^3 + 2*n0*n3*d1/dm^3 + 2*n0*n2*d2/dm^3 + ... %
  2*n1^2*d2/dm^3 + 2*n0*n2*d2/dm^3+2*n0*n1*d3/dm^3+ ... %
  6*n1^2*d1^2/dm^4 + 6*n0*n2*d1^2/dm^4+6*n0*n1^2*d1*d2/dm^4 + ...
%
  w2_psi*d1/dm + w2*d2/dm + ... %
  w1_psi*d2/dm + w1*d3/dm+ ... %
  w1_psi*d1^2/dm^2 + w1^2*d1*d2/dm^2 + ... %
  w1_psi*d2/dm + w1*d3/dm + ... %
  w0_psi*d3/dm + w0*d4/dm + ... %

```

```

w0_psi*d1*d2/dm^2 + w0*d2*d2/dm^2 + w0*d1*d3/dm^2 + ... %
w1_psi*d1^2/dm^2 + w1^2*d1*d2/dm^2 + ... %
2*w0_psi*d1*d2/dm^2 + 2*w0*d2*d2/dm^2 + 2*w0*d1*d3/dm^2 + ...%
2*w0_psi*d1^3/dm^3 + 2*w0^3*d1^2*d2/dm^3 ... %
);

xi2 = xi*xi;
xi3 = xi2*xi;

% derivatives of v with respect to x
v1 = w1+ xi*w0;
v2 = w2+2*xi*w1+xi2*w0;
v3 = w3+3*xi*w2+3*xi2*w1+xi3*w0;

% derivatives with respect to psi
v1_psi = w1_psi+xi_der*w0+xi*w0_psi;
v2_psi = w2_psi+2*xi_der*w1+2*xi*w1_psi+2*xi*xi_der*w0+xi2*w0_psi;
v3_psi = w3_psi+3*xi_der*w2+3*xi*w2_psi+6*xi*xi_der*w1...
        +3*xi2*w1_psi + 3*xi2*xi_der*w0+xi3*w0_psi;

% bound on functions to interpoate
f1 = v1*v2;
f2 = v3*v2;
g = w0*v1;

% bound on derivative with respect to psi of
% functions to interpolate
f1_psi = v1_psi*v2+v1*v2_psi;
f2_psi = v3_psi*v2+v3*v2_psi;
g_psi = w0_psi*v1+w0*v1_psi;

% here = f1(round(0.5*length(f1)))

% bound on all sub bounds
out = nm(max(sup([f1,f2,g,f1_psi,f2_psi,g_psi])));

```


3.4 bound_numer_unstable.m

```
function [M_psi,M_x,M_q] = bound_numer_unstable(dm,rho_x,rho_q,a_q,b_q)

%
% constants
%

a_psi = nm(1);
b_psi = nm(1);
rho_psi = 1;
ntilde = 1;

min_abs_q = (a_q+b_q)/2 -(b_q-a_q)*(rho_q+1/rho_q)/4;

%
% bound for interpolation in psi -----
%

% top of ellipse  $E_{\rho_\psi}$ 
max_imag_x = 0;
max_real_psi = 1;
max_abs_q = sup(real(b_q));
prodq = abs(log(a_q)/2);

max_xi = bound_xi(a_psi,b_psi,rho_psi,a_q,b_q,1,ntilde);
max_xi_der = bound_xi_der...
    (max_abs_q,min_abs_q,a_q,b_q,max_real_psi,rho_q,rho_psi,a_psi,b_psi,ntilde);

M_psi = 2*local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);

%
% bound for interpolation in x -----
%

% top of ellipse  $E_{\rho_\beta}$ 
```

```

max_imag_x = (rho_x-1/rho_x)/2;
max_real_psi = 1;
% max_abs_q the same
% prodq the same
% max_xi the same
% max_xi_der the same

M_x = local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);

%
% bound for interpolation in q
%

% top of ellipse  $E_{\rho_\beta}$ 
max_imag_x = 0;
max_real_psi = 1;
max_abs_q = (a_q+b_q)/2 + ((b_q-a_q)/4)*(rho_q+1/rho_q);

max_xi = bound_xi(a_psi,b_psi,rho_psi,a_q,b_q,1,ntilde);
max_xi_der = bound_xi_der...
    (max_abs_q,min_abs_q,a_q,b_q,max_real_psi,rho_q,rho_psi,a_psi,b_psi,ntilde);

min_abs_q = (a_q+b_q)/2 - (b_q-a_q)*(rho_q+1/rho_q)/4;
prodq = abs(log(min_abs_q)/2);

M_q = 2*local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,max_xi,max_xi_der,prodq);

%
% local_bounds
%

function out = local_bounds(dm,max_imag_x,max_real_psi,max_abs_q,xi,xi_der,prodq)

pie = nm('pi');
prod = pie/2;

```

```

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
n0 = temp(1);
n1 = temp(2); % first derivative
n2 = temp(3); % second derivative
n3 = temp(4); % third derivative
n4 = temp(5);

```

```

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
max_real_psi = 0;
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
d1 = temp(2); % first derivative
d2 = temp(3); % second derivative
d3 = temp(4); % third derivative
d4 = temp(5);

```

```

% bound on  $w(x)$ , the conjugate of  $w(x)$ , and their derivatives
% on the ellipse  $E_{\rho_x}$ 

```

```

%  $w(x)$ 
w0 = n0^2/dm^2;

```

```

%  $w'(x)$ 
w1 = 2*prod*( n0*n1/dm^2 + w0*d1/dm );

```

```

%  $w''(x)$ 
w2 = 4*prod^2*(n1^2/dm^2+n0*n2/dm^2+2*n0*n1*d1/dm^3+w1*d1/dm+w0*d2/dm+w0*d1^2/dm^2);

```

```

%  $w'''(x)$ 
w3 = 8*prod^3*( 2*n1*n2/dm^2+2*n1^2*d1/dm^3+n1*n2/dm^2+n0*n3/dm^2+2*n0*n2*d1/dm^3+...
    2*n1^2*d1/dm^3+2*n0*n2*d1/dm^3+2*n0*n1*d2/dm^3+6*n0*n1*d1^2/dm^4+w2*d1/dm-
    + w1*d2/dm+...
    w1*d1^2/dm^2+w1*d2/dm+w0*d3/dm+w0*d1*d2/dm^2+w1*d1^2/dm^2+2*w0*d1*d2/dm^2+...
    2*w0*d1^3/dm^3);

```

```

% bound on derivative of  $w(x)$  with respect to  $\psi$ 
w0_psi = 2*prodq*(n0*n1/dm^2);

```

```

% derivative of w'(x) with respect to psi
w1_psi = 2*prodq^2*(n0*n2/dm^2+n1^2/dm^2+w0_psi*d1/dm);

% bound on derivative of w''(x) with respect to psi
% —w''(x)— < 4*prod^2*( (2*n0*n1*d1)/dm^3+(n1^2+n0*n2+w0*d1^2)/dm^2-
+ (w1*d1+ w0*d2)/dm );
w2_psi = 4*prodq^3*( (2*n1^2*d1+2*n0*n2*d1+2*n0*n1*d2)/dm^3 + ...
(2*n1*n2+n1*n2+n0*n3+w0_psi*d1^2+ w0^2*d1*d2)/dm^2 + ...
(w1_psi*d1+w1*d2+w0_psi*d2+w0*d3)/dm);

% bound on derivative of w'''(x) with respect to psi
w3_psi = 8*prodq^4*(...
2*n2*n2/dm^2+2*n1*n3/dm^2 + ... %
4*n1*n2*d1/dm^3+2*n1^2*d2/dm^3+ ...%
n2^2/dm^2+ n1*n3/dm^2 + ... %
n1*n3/dm^2+n0*n4/dm^2 + ... %
2*n1*n2*d1/dm^3 + 2*n0*n3*d1/dm^3 + 2*n0*n2*d2/dm^3 +... %
4*n1*n2*d1/dm^3 + 2*n1^2*d2/dm^3 + ... %
2*n1*n2*d1/dm^3 + 2*n0*n3*d1/dm^3 + 2*n0*n2*d2/dm^3 + ... %
2*n1^2*d2/dm^3 + 2*n0*n2*d2/dm^3+2*n0*n1*d3/dm^3+ ... %
6*n1^2*d1^2/dm^4 + 6*n0*n2*d1^2/dm^4+6*n0*n1^2*d1*d2/dm^4 + ...
%
w2_psi*d1/dm + w2*d2/dm +... %
w1_psi*d2/dm + w1*d3/dm+ ... %
w1_psi*d1^2/dm^2 + w1^2*d1*d2/dm^2 + ... %
w1_psi*d2/dm + w1*d3/dm + ... %
w0_psi*d3/dm + w0*d4/dm + ... %
w0_psi*d1*d2/dm^2 + w0*d2*d2/dm^2 + w0*d1*d3/dm^2 + ... %
w1_psi*d1^2/dm^2 + w1^2*d1*d2/dm^2 + ... %
2*w0_psi*d1*d2/dm^2 + 2*w0*d2*d2/dm^2 + 2*w0*d1*d3/dm^2 + ...%
2*w0_psi*d1^3/dm^3 + 2*w0^3*d1^2*d2/dm^3 ... %
);

xi2 = xi*xi;
xi3 = xi2*xi;

% derivatives of v with respect to x
v1 = w1+ xi*w0;
v2 = w2+2*xi*w1+xi2*w0;
v3 = w3+3*xi*w2+3*xi2*w1+xi3*w0;

```

```

% derivatives with respect to psi
v1_psi = w1_psi+xi_der*w0+xi*w0_psi;
v2_psi = w2_psi+2*xi_der*w1+2*xi*w1_psi+2*xi*xi_der*w0+xi2*w0_psi;
v3_psi = w3_psi+3*xi_der*w2+3*xi*w2_psi+6*xi*xi_der*w1...
        +3*xi2*w1_psi + 3*xi2*xi_der*w0+xi3*w0_psi;

% bound on functions to interpolate
f1 = v1*v2;
f2 = v3*v2;
g = w0*v1;

% bound on derivative with respect to psi of
% functions to interpolate
f1_psi = v1_psi*v2+v1*v2_psi;
f2_psi = v3_psi*v2+v3*v2_psi;
g_psi = w0_psi*v1+w0*v1_psi;

% bound on all sub bounds
out = nm(max(sup([f1,f2,g,f1_psi,f2_psi,g_psi])));

```

3.5 bound_sub_integrals.m

```

function [M_psi,M_x,M_q] = bound_sub_integrals(dm,rho_psi,rho_x,rho_q,a_q,b_q)

% constants
pie = nm('pi');
prod = pie/2;

% -----
% bound for interpolation in psi
% -----

% top of ellipse  $E_{\rho_x}$ 
max_imag_x = 0;
max_real_psi = (1+(rho_psi+1/rho_psi)/2)/2;

```

```

max_abs_q = sup(real(b_q));

vec = bound(prod,dm,max_imag_x,max_real_psi,max_abs_q);

% largest bound on all the sub integrands when alpha = i beta
M_psi = sup(2*nm(max(vec)));

% -----
% bound for interpolation in x
% -----

% top of ellipse  $E_{\rho_x}$ 
max_imag_x = (rho_x-1/rho_x)/2;
max_real_psi = 1;
max_abs_q = sup(real(b_q));

vec = bound(prod,dm,max_imag_x,max_real_psi,max_abs_q);

% largest bound on all the sub integrands when alpha = i beta
M_x = max(vec);

% -----
% bound for interpolation in q
% -----

% top of ellipse  $E_{\rho_x}$ 
max_imag_x = 0;
max_real_psi = 1;
max_abs_q = (a_q+b_q)/2 + ((b_q-a_q)/4)*(rho_q+1/rho_q);

vec = bound(prod,dm,max_imag_x,max_real_psi,max_abs_q);

% largest bound on all the sub integrands when alpha = i beta
M_q = sup(2*nm(max(vec)));

```

```

% -----
% function for bounds
% -----
function vec = bound(prod,dm,max_imag_x,max_real_psi,max_abs_q)

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega'))$ 
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);

n0 = temp(1);
n1 = temp(2); % first derivative
n2 = temp(3); % second derivative
n3 = temp(4); % third derivative

% bound on  $\vartheta_1^{(m)}(\frac{\pi}{2\omega}(\omega x \pm i\omega' + n\omega + i\beta))$ 
max_real_psi = 0;
temp = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,4);
d1 = temp(2); % first derivative
d2 = temp(3); % second derivative
d3 = temp(4); % third derivative

% w(x)
w0 = n0^2/dm^2;

% w'(x)
w1 = 2*prod*( n0*n1/dm^2 + w0*d1/dm );

% w''(x)
w2 = 4*prod^2*(n1^2/dm^2+n0*n2/dm^2+2*n0*n1*d1/dm^3+w1*d1/dm+w0*d2/dm+w0*d1^2/dm^2);

% w'''(x)
w3 = 8*prod^3*( 2*n1*n2/dm^2+2*n1^2*d1/dm^3+n1*n2/dm^2+n0*n3/dm^2+2*n0*n2*d1/dm^3+...
    2*n1^2*d1/dm^3+2*n0*n2*d1/dm^3+2*n0*n1*d2/dm^3+6*n0*n1*d1^2/dm^4+w2*d1/dm-
    + w1*d2/dm+...
    w1*d1^2/dm^2+w1*d2/dm+w0*d3/dm+w0*d1*d2/dm^2+w1*d1^2/dm^2+2*w0*d1*d2/dm^2+...
    2*w0*d1^3/dm^3);

% bounds on the 10 sub integrands

```

```

vec = [ w1*w2; ...
        w0*w2+2*w1*w1; ...
        w1*w0+2*w0*w1; ...
        w0*w0; ...
        w3*w2; ...
        3*w2*w2+2*w3*w1;
        w3*w0+6*w2*w1+3*w1*w2;
        3*w2*w0+6*w1*w1+w0*w2;
        3*w1*w0+2*w0*w1;
        w0*w0];

% bound on 10 sub integrands
vec = sup(vec);

```

3.6 bound_theta1_m.m

```
function out = bound_theta1_m(max_imag_x,max_real_psi,max_abs_q,m)
```

The first Jacobi Theta function is given by the series,

$$\vartheta_1(z) = 2 \sum_{n=0}^{\infty} (-1)^n q^{(n+1/2)^2} \sin((2n+1)z),$$

and its m th derivative is given by,

$$\vartheta_1^{(m)}(z) = 2 \sum_{n=0}^{\infty} (-1)^n q^{(n+1/2)^2} (2n+1)^m f((2n+1)z),$$

where $f(\cdot) = \sin(\cdot)$ if $m \equiv 0 \pmod{4}$, $f(\cdot) = \cos(\cdot)$ if $m \equiv 1 \pmod{4}$, $f(\cdot) = -\sin(\cdot)$ if $m \equiv 2 \pmod{4}$, and $f(\cdot) = -\cos(\cdot)$ if $m \equiv 3 \pmod{4}$.

Now for $m \geq 0$,

$$\begin{aligned} \left| (-1)^n q^{(n+1/2)^2} (2n+1)^m f((2n+1)z) \right| &\leq \left| q^{(n+1/2)^2} (2n+1)^m e^{(2n+1)|\Im(z)|} \right| \\ &\leq \left| q^{N^2+1/2} e^{(2N+1)|\Im(z)|} (2N+1)^m \right| q^n, \end{aligned}$$

for $n \geq N$ with N sufficiently large that

$$\left| q^{n^2+1/4} e^{(2n+1)|\Im(z)|} (2n+1)^m \right| \leq \left| q^{N^2+1/4} e^{(2N+1)|\Im(z)|} (2N+1)^m \right|,$$

whenever $n \geq N$. To determine how large N must be, we define,

$$g(x) := q^{x^2+1/4} e^{(2x+1)|\Im(z)|} (2x+1)^m.$$

We will take N large enough that $g'(x) < 0$ whenever $x \geq N$. If $m = 0$, then

$$g'(x) = 2q^{x^2+1/4} e^{(2x+1)|\Im(z)|} (x \log(q) + |\Im(z)|),$$

and we see that

$$N > -\frac{|\Im(z)|}{\log(q)}$$

suffices. If $M > 0$,

$$g'(x) = 2q^{x^2+1/4} e^{(2x+1)|\Im(z)|} (2x+1)^{m-1} ((x \log(q) + |\Im(z)|)(2x+1) + m).$$

From

$$(x \log(q) + |\Im(z)|)(2x+1) + m < 0,$$

we find that

$$N > -\frac{2|\Im(z)| + \log(q) + \sqrt{(2|\Im(z)| + \log(q))^2 - 8 \log(q)(|\Im(z)| + m)}}{4 \log(q)}$$

suffices. For such an N , the error of the summation truncation is

$$q^{N^2+1/4} e^{(2N+1)|\Im(z)|} (2N+1)^m \frac{q^N}{1-q}.$$

%% constants

pie = nm('pi');

q = nm(max_abs_q);

% Find max of $\vartheta_1(\frac{\pi}{2\omega}(\omega x \pm i\omega' + n\omega + i\beta))$ on the ellipse E_ρ .

% abz = pie*(omega*max_imag_x+omega_prime+max_real_beta)/(2*omega);

abz = sup(pie*max_imag_x + abs(log(max_abs_q))*(1+max_real_psi))/2;

```

qlog = log(q);
one_fourth = nm(1)/4;
one_half = nm(1)/2;
con = q^one_fourth*exp(abz);

%%

% find N large enough that
%  $f(x) := q^{x^2+1/4}e^{(2x+1)|\Im(z)|}(2x+1)^m$ 
% is decreasing for  $x \geq N$ .
if m == 0
    N = ceil(sup(-abz/qlog));
else
    c1 = 2*abz+qlog;
    disc = c1^2-8*qlog*(abz+m);
    c2 = -c1/(4*qlog);
    c3 = disc/(4*qlog);
    N = ceil(sup(c2-c3));
end

% compute theta(z,q) and its derivatives with partial sum
%  $\vartheta_1^{(m)}(z) = 2 \sum_{n=0}^{N-1} (-1)^n q^{(n+1/2)^2} (2n+1)^m f((2n+1)z)$ 
out = nm(zeros(m+1,1));
for n = 0:N-1
    prod = 2*n+1;
    zprod = prod*abz;
    gen = exp(zprod + log(q)*(n+one_half)^2);

    for ind = 1:m+1
        out(ind) = out(ind) + gen*prod^(ind-1);
    end
end

% truncation error
%  $q^{N^2+1/4}e^{(2N+1)|\Im(z)|}(2N+1)^m \frac{q^N}{1-q}$ 
% note that con =  $q^{1/4}e^{|\Im(z)|}$ 

for ind = 0:m

```

```

    out(ind+1) = out(ind+1) + con*exp(N^2*log(q)+2*abz*N)*(2*N+1)^ind*q^N/(1-
q);
end

```

```

out = 2*out;
out = sup(out);

```

3.7 bound_xi.m

```
function out = bound_xi(a_psi,b_psi,rho_psi,a_q,b_q,rho_q,ntilde)
```

```
% latex description
```

Now

$$\begin{aligned}\xi(\alpha) &= 2i \left(\zeta(\alpha) - \frac{\alpha}{\omega} \zeta(\omega) \right) \\ &= 2i \left(\frac{\pi}{2\omega} \cot \left(\frac{\pi\alpha}{2\omega} \right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1-q^{2k}} \sin \left(\frac{k\pi\alpha}{\omega} \right) \right).\end{aligned}$$

When $\alpha = \omega + i\psi\omega'$ we have

$$\begin{aligned}\xi(\omega + i\psi\omega') &= 2i \left(\frac{\pi}{2\omega} \cot \left(\frac{\pi(\omega + i\psi\omega')}{2\omega} \right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1-q^{2k}} \sin \left(\frac{k\pi(\omega + i\psi\omega')}{\omega} \right) \right) \\ &= 2i \left(-\frac{\pi}{2\omega} \tan \left(\frac{i\pi\psi\omega'}{2\omega} \right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1-q^{2k}} (-1)^k \sin \left(\frac{ik\pi\psi\omega'}{\omega} \right) \right).\end{aligned}\tag{3.1}$$

When $\alpha = i\psi\omega'$ we have,

$$\xi(i\psi\omega') = 2i \left(\frac{\pi}{2\omega} \cot \left(\frac{\pi(i\psi\omega')}{2\omega} \right) + \frac{2\pi}{\omega} \sum_{k=1}^{\infty} \frac{q^{2k}}{1-q^{2k}} \sin \left(\frac{k\pi(i\psi\omega')}{\omega} \right) \right).$$

Note that if either $\psi \in [0, 1]$ and $0 < |q| < 1$ with $q \in \mathbb{C}$, or if $q \in [q_a, q_b] \subset (0, 1)$, $|\Re(\psi)| < 2$ and $|\Im(\psi)| < \frac{\pi}{|\log(q_a)|}$, then (3.2) is analytic in that region. If either $\psi \in (0, 1]$ and $0 < |q| < 1$ with $q \in \mathbb{C}$, or if $q \in [q_a, q_b] \subset (0, 1)$, $|\Re(\psi)| < 2$ and $|\psi| > 0$, then (3.3) is analytic.

Note that $\sin\left(\frac{i\pi\psi\omega'k}{\omega}\right) = \frac{1}{2i} (q^{\psi k} - q^{-\psi k})$ so that in both cases the infinite sum is bounded by

$$\begin{aligned} 2 \sum_{k=1}^{\infty} \frac{q_0^{\gamma k}}{1 - q_0^{2k}} &\leq \frac{2}{1 - q_0^2} \sum_{k=1}^{\infty} q_0^{\gamma k} \\ &\leq \frac{2q_0^{\gamma}}{(1 - q_0^2)(1 - q_0^{\gamma})}. \end{aligned}$$

where $|q| \leq q_0 < 1$ and $\gamma := 2 - |\Re(\psi)|$.

Let M be a bound on $\tan\left(\frac{-i\psi \log(q)}{2}\right)$ or $\cot\left(\frac{-i\psi \log(q)}{2}\right)$, depending on the choice of α . Then

$$|\omega\xi(\tilde{n} + i\psi\omega')| \leq M\pi + \frac{8\pi q_0^{\gamma}}{(1 - q_0^2)(1 - q_0^{\gamma})}.$$

%%

pie = nm('pi');

%
% get bound on tan or cot
%

pts = 300;
theta = nm(linspace(0,sup(2*pie),pts));
if rho_psi == 1
 psivec = nm(linspace(inf(a_psi),sup(b_psi),pts));
else
 psivec = nm(1)/2+ (rho_psi*exp(1i*theta)+exp(-1i*theta)/rho_psi)/4;
end

if rho_q == 1
 qvec = nm(linspace(inf(a_q),sup(b_q),pts));
else

```

    qvec = (a_q+b_q)/2+(b_q-a_q)*(rho_q*exp(1i*theta)+exp(-1i*theta)/rho_q)/4;
end

qv = nm(qvec(1:end-1),qvec(2:end));
psiv = nm(psivec(1:end-1),psivec(2:end));

% temp = sup(abs(tan(-1i*psiv.*log(qv)/2)))

if ntilde == 1
    % get bound on tan(1i*pi*psi*omega_prime/(2*omega))
    temp = sup(abs(tan(-1i*psiv.*log(qv)/2)));
elseif ntilde == 0
    % get bound on cot(1i*pi*psi*omega_prime/(2*omega))
    temp = sup(abs(cot(-1i*psiv.*log(qv)/2)));
end

if sum(sum(isnan(temp))) > 0
    error('NaN present');
end

M = max(max(temp));

max_abs_real_psi = (a_psi + b_psi)/2+(b_psi-a_psi)*(rho_psi +1/rho_psi)/4;
max_abs_q = (a_q + b_q)/2+(b_q-a_q)*(rho_q +1/rho_q)/4;

gamma = 2-max_abs_real_psi;
qg = max_abs_q^gamma;
out = pie*M+ (8*pie*qg)/((1-max_abs_q^2)*(1-qg));

```

3.8 bound_xi_der.m

`function out = bound_xi_der(abs_q,min_abs_q,a,b,max_real_psi,rho_q,rho_psi,a_psi,b_psi,ntilde)`

`% latex comments`

If $\tilde{n} = 1$, then from the q -series representation of the Weierstrass elliptic function,

$$\begin{aligned} \frac{\partial}{\partial \psi} \xi(\tilde{n}\omega + i\psi\omega') &= 2\omega' \left(\wp(\tilde{\omega} + i\psi\omega') + \frac{\zeta(\omega)}{\omega} \right) \\ &= 2\omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \sec^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} \cos \left(\frac{ik\pi\psi\omega'}{\omega} \right) \right) \\ &= 2\omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \sec^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} \left(\frac{q^{\psi k} + q^{-\psi k}}{2} \right) \right). \end{aligned} \quad (3.2)$$

If $\tilde{n} = 0$, then from the q -series representation of the Weierstrass elliptic function,

$$\begin{aligned} \frac{\partial}{\partial \psi} \xi(\tilde{n}\omega + i\psi\omega') &= 2\omega' \left(\wp(\tilde{\omega} + i\psi\omega') + \frac{\zeta(\omega)}{\omega} \right) \\ &= 2\omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \csc^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} \frac{kq^{2k}}{1-q^{2k}} \cos \left(\frac{ik\pi\psi\omega'}{\omega} \right) \right) \\ &= 2\omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \csc^2 \left(\frac{i\pi\psi\omega'}{2\omega} \right) - \frac{2\pi^2}{\omega^2} \sum_{k=1}^{\infty} \frac{kq^{2k}}{1-q^{2k}} \left(\frac{q^{\psi k} - q^{-\psi k}}{2i} \right) \right). \end{aligned} \quad (3.3)$$

Note that if either $\psi \in [0, 1]$ is fixed and $|q| < 1$ with $q \in \mathbb{C}$, or if $q \in [q_a, q_b] \subset$

$(0, 1)$ is fixed and $|\Im(\psi)| < \frac{\pi}{|\log(q_a)|}$, then (3.2) is analytic in that region. Note that if either $\psi \in [0, 1]$ is fixed and $|q| < 1$ with $q \in \mathbb{C}$, or if $q \in [q_a, q_b] \subset (0, 1)$ is fixed and $\Re(\psi) > 0$, then (3.2) is analytic in that region.

Let $0 \leq q_0 < 1$, $\gamma \in \mathbb{R}$, and define

$$\begin{aligned} f(x) &:= \sum_{k=0}^{\infty} \frac{1}{\gamma \log(q_0)} q_0^{\gamma k x} \\ &= \frac{1}{\gamma \log(q_0)} \frac{1}{1 - q_0^{\gamma x}}. \end{aligned} \tag{3.4}$$

Note that

$$\begin{aligned} f'(x) &= \sum_{k=0}^{\infty} k q_0^{\gamma k x} \\ &= \frac{q_0^{\gamma x}}{(1 - q_0^{\gamma x})^2}. \end{aligned} \tag{3.5}$$

Then

$$\begin{aligned} \left| \sum_{k=1}^{\infty} (-1)^{\tilde{n}k} \frac{k q^{2k}}{1 - q^{2k}} (q^{\psi k} + q^{-\psi k}) \right| &\leq 2 \sum_{k=0}^{\infty} \frac{k q_0^{\gamma k}}{1 - q_0^{2k}} \\ &\leq \frac{2}{1 - q_0^2} \sum_{k=0}^{\infty} k q_0^{\gamma k} \\ &\leq \frac{2}{1 - q_0^2} \frac{q_0^{\gamma}}{(1 - q_0^{\gamma})^2}, \end{aligned} \tag{3.6}$$

where $|q| \leq q_0 < 1$ and $\gamma := 2 - \Re(\psi)$.

%%

%
% constants
%

pie = nm('pi');
gamma = 2-max_real_psi;
qg = abs_q^gamma;

```

% error check
if inf(gamma) <= 0
    error('max_psi too big');
end

%
% get bound on (sec/csc)(1i*pi*psi*omega_prime/(2*omega))
%

pnts = 500;
% bound on stadium in variable psi
qvec = nm(linspace(inf(a),sup(b),pnts));
theta = nm(linspace(0,sup(2*pie),pnts));
psivec = (a_psi+b_psi)/2 + (b_psi-a_psi)*(rho_psi*exp(1i*theta)+exp(-1i*theta)/rho_psi)/4;
qv = nm(qvec(1:end-1),qvec(2:end));
psiv = nm(psivec(1:end-1),psivec(2:end));
if ntilde == 1
    temp = sup(abs(sec(-1i*psiv.*log(qv)/2)));
elseif ntilde == 0
    temp = sup(abs(csc(-1i*psiv.*log(qv)/2)));
end
if sum(sum(isnan(temp))) > 0
    error('NaN present');
end
max_temp1 = max(max(temp));

% bound on stadium in variable q
psivec = nm(linspace(a_psi,b_psi,pnts));
theta = nm(linspace(0,sup(2*pie),pnts));
qvec = (a+b)/2 + (b-a)*(rho_q*exp(1i*theta)+exp(-1i*theta)/rho_q)/4;
qv = nm(qvec(1:end-1),qvec(2:end));
psiv = nm(psivec(1:end-1),psivec(2:end));

if ntilde == 1
    temp = sup(abs(sec(-1i*psiv.*log(qv)/2)));
elseif ntilde == 0
    temp = sup(abs(csc(-1i*psiv.*log(qv)/2)));
end
if sum(sum(isnan(temp))) > 0
    error('NaN present');
end
max_temp2 = max(max(temp));

```



```

% take maximum of two bounds
temp_bd = nm(max(max_temp1,max_temp2));

%
% bound on infinite sum part
%

out = (2/(1-abs_q^2))*(qg/(1-qg)^2);

%
% combine all parts
%

out = 2*abs(log(min_abs_q))*pie*(out + temp_bd^2/4);

```

3.9 cf_biv_cheby.m

```

function cf = cf_biv_cheby(m,n,num_funs,fun)
% function cf_padau(m,n,fun)
%

```

```
% Returns the coefficients for bivariate Chebyshev interpolation
```

Let $f(x, y) : [-1, 1] \times [-1, 1] \rightarrow \mathbb{C}$. Let $T_i(x)$, $T_j(y)$ be Chebyshev polynomials of the first kind. This function returns a matrix `cf` of coefficients to the bivariate interpolation polynomial, $p(x, y) = \sum_{i=0}^m \sum_{j=0}^n c_{i,j} T_i(x) T_j(y)$, where $c_{i,j}$ is the i th + 1, j th + 1 row and column of the matrix `cf`.

```
% constants
```

```
pie = nm('pi');
c_1 = pie/(2*(m+1));
c_2 = pie/(2*(n+1));
```

```
% theta for x and y
```

```
theta_xr = c_1*(2*(0:1:m)+1);
theta_ys = c_2*(2*(0:1:n)+1);
```

```
% x and y interpolation points
```

```
xr = cos(theta_xr);
ys = cos(theta_ys);
```

```
% evaluate function at grid points
```

```
f_xy = nm(zeros(m+1,n+1,num_funs));
```

```
fun2 = (x)(fun(x,ys));
```

```
for r = 1:m+1
```

```
    %
    % get function values
    %
```

```
    f_xy(r,,:) = fun2(xr(r));
```

```
end
```

```
% get Chebyshev polynomials evaluated at points;
```

```

Tx = cos((0:1:m).'*theta_xr);
Ty = cos(theta_ys.'*(0:1:n));

cf = nm(zeros(m+1,n+1,num_funs));

for j = 1:num_funs
    cf(:,j) = (4/((m+1)*(n+1)))*Tx*f_xy(:,j)*Ty;
    cf(:,1,j) = cf(:,1,j)/2;
    cf(1,:,j) = cf(1,:,j)/2;
end

```

3.10 cf_eval.m

```

function out = cf_eval(cf,x,y)
% function out = cf_eval(cf,x,y)
%
% Evaluates the two dimensional Chebyshev polynomial at (x,y)

```

Let $f(x, y) : [-1, 1] \times [-1, 1] \rightarrow \mathbb{C}$. Let $T_i(x)$, $T_j(y)$ be Chebyshev polynomials of the first kind. This function evaluates $p(x, y) = \sum_{i=0}^m \sum_{j=0}^n c_{i,j} T_i(x) T_j(y)$, where $c_{i,j}$ is the i th + 1, j th + 1 row and column of the matrix cf.

```

m = size(cf,1);
n = size(cf,2);
theta_x = acos(x);
theta_y = acos(y);
ind_x = 0:1:(m-1);
ind_y = 0:1:(n-1);

```

```

Tx = cos(ind_x.*theta_x);
Ty = cos(theta_y.*ind_y);

out = nm(zeros(length(x),length(y)));
for j = 1:length(x)
    for k = 1:length(y)
        out(j,k) = sum(sum(cf.*(Tx(:,j)*Ty(k,:))));
    end
end
end

```

3.11 cf_eval_theta.m

```

function out = cf_eval_theta(cf,theta_x,theta_y)
% function out = cf_eval(cf,x,y)
%
% Evaluates the two dimensional Chebyshev polynomial at (x,y)

```

Let $f(x, y) : [-1, 1] \times [-1, 1] \rightarrow \mathbb{C}$. Let $T_i(x)$, $T_j(y)$ be Chebyshev polynomials of the first kind. This function evaluates $p(x, y) = \sum_{i=0}^m \sum_{j=0}^n c_{i,j} T_i(x) T_j(y)$, where $c_{i,j}$ is the i th + 1, j th + 1 row and column of the matrix cf.

```

m = size(cf,1);
n = size(cf,2);
% theta_x = acos(x);
% theta_y = acos(y);
ind_x = 0:1:(m-1);
ind_y = 0:1:(n-1);

```

```

Tx = cos(ind_x.*theta_x);
Ty = cos(theta_y.*ind_y);

```

```

out = nm(zeros(length(theta_x),length(theta_y)));
for j = 1:length(theta_x)

```

```

    for k = 1:length(theta_y)
        out(j,k) =sum(sum(cf.*(Tx(:,j)*Ty(k,:))));
    end
end
end

```

3.12 cheby_eval.m

```

function out = cheby_eval(a,ltx,rtx,lty,rty,ints_x,ints_y)
% cheby_taylor(a,ltx,rtx,lty,rty,ints_x,ints_y)
%
% a = Chebyshev coefficients
% ltx = left theta value for variable x
% rtx = right theta value for variable x
% lty = left theta value for variable y
% rty = right theta value for variable y
% ints_x = number of intervals in variable x
% ints_y = number of intervals in variable y

Nx = size(a,1)-1; % degree of polynomial in x
Ny = size(a,2)-1; % degree of polynomial in y
tx = linspace(ltx,rtx,ints_x+1).'; % theta points in x
ty = linspace(lty,rty,ints_y+1).'; % theta points in y
indx = 0:1:Nx;
indy = (0:1:Ny).';

zx = nm(tx)*indx; % function argument for x
zy = indy*nm(ty); % function argument for y

% cos and sin
cosx = cos(zx);
cosy = cos(zy);

out = cosx*a*cosy;

```

3.13 cheby_taylor.m

```
function out = cheby_taylor(a,ltx,rtx,lty,rty,ints_x,ints_y)
% cheby_taylor(a,ltx,rtx,lty,rty,ints_x,ints_y)
%
% a = Chebyshev coefficients
% ltx = left theta value for variable x
% rtx = right theta value for variable x
% lty = left theta value for variable y
% rty = right theta value for variable y
% ints_x = number of intervals in variable x
% ints_y = number of intervals in variable y

Nx = size(a,1)-1; % degree of polynomial in x
Ny = size(a,2)-1; % degree of polynomial in y
tx = linspace(ltx,rtx,ints_x+1).'; % theta points in x
ty = linspace(lty,rty,ints_y+1).'; % theta points in y
indx = 0:1:Nx;
indy = (0:1:Ny).';

zx = nm(tx(1:end-1))*indx; % function argument for x
zy = indy*nm(ty(1:end-1)); % function argument for y

% function argument for error
ztx = nm(tx(1:end-1),tx(2:end))*indx;
zty = indy*nm(ty(1:end-1),ty(2:end));

% cos and sin
cosx = cos(zx);
cosy = cos(zy);
sinx = sin(zx);
```

```

siny = sin(zy);

% find delta theta_x and delta theta_y
txint = nm(tx);
tyint = nm(ty);
hx = nm(0,max(sup(txint(2:end)-txint(1:end-1))));
hy = nm(0,max(sup(tyint(2:end)-tyint(1:end-1))));

indx = repmat(indx,length(tx)-1,1);
indy = repmat(indy,1,length(ty)-1);

dx1 = -(indx.*sinx);
dx2 = -(indx.^2.*cosx);
dx3 = (indx.^3.*sinx);
dx4 = (indx.^4.*cosx);

dy1 = -(indy.*siny);
dy2 = -(indy.^2.*cosy);
dy3 = (indy.^3.*siny);
dy4 = (indy.^4.*cosy);

% Taylor expansion in two variables
out = cosx*a*cosy ...
    + dx1*a*cosy*hx+cosx*a*dy1*hy ...
    + ( dx2*a*cosy*hx^2+2*dx1*a*dy1*hx*hy+cosx*a*dy2*hy^2 )/2 ...
    + ( dx3*a*cosy*hx^3 + 3*dx2*a*dy1*hx^2*hy + 3*dx1*a*dy2*hx*hy^2+cosx*a*dy3*hy^3-
    )/6 ...
    + ( dx4*a*cosy*hx^4+4*dx3*a*dy1*hx^3*hy+6*dx2*a*dy2*hx^2*hy^2+
    ...
    4*dx1*a*dy3*hx*hy^3 + cosx*a*dy4*hy^4 )/24;

% for error term
costx = cos(ztx);
costy = cos(zty);
sintx = sin(ztx);
sinty = sin(zty);

% for error term
ft_1 = -hx^5*(indx.^5.*sintx)*a*costy/120;
ft_2 = -hx^4*hy*(indx.^4.*costx)*a*(indy.*sinty)/24;
ft_3 = -hx^3*hy^2*(indx.^3.*sintx)*a*(indy.^2.*costy)/12;

```

```

ft_4 = -hx^2*hy^3*(indx.^2.*costx)*a*(indy.^3.*sinty)/12;
ft_5 = -hx*hy^4*(indx.*sintx)*a*(indy.^4.*costy)/24;
ft_6 = -hy^5*costx*a*(indy.^5.*sinty)/120;

% find err interval
err = ft_1+ft_2+ft_3+ft_4+ft_5+ft_6;

% combine partial sum and error interval
out = out + err;

```

3.14 check_alpha_distinct.m

```

function success = check_alpha_distinct(k)

success = 1;

K = elliptic_integral(k,1);
E = elliptic_integral(k,2);
k2 = k*k;
c1 = 1-k2;
b1 = k2*K/(E-K);
b2 = k2*c1*K/(c1*K-E);
b3 = c1*K/E;

if sup(b1) >= inf(b2)
    success = 0;
    return
end

if sup(b2) >= inf(b3)
    success = 0;
    return
end

```


3.15 create_tableA.m

```
% create Table of lower instability for the paper
clear all; clc; curr_dir = local_startup;
```

```
file_name = 'instability_result_lower.mat';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
```

```
fprintf('\n\n\\begin{table}[!b]\\n\\begin{tabular}{c c c c c c c c c c}\\n\\hline');  
fprintf('nk_L&k_R&q_L&q_R&M_x&M_q&n\rho_q&N_x&N_q&M_n\\lambda\\\\\\');
```

```
for j = 1:length(d.RK)
```

```

fprintf('\n\\hline\n');
fprintf('%5.5g',d.KL(j));
fprintf(' & ');
fprintf('%5.5g',d.RK(j));
fprintf(' & ');

file_name = ['d_lower_unstable_',num2str(j),'.mat'];
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
r = ld.var;

fprintf('%5.5g',sup(r.q_min));
fprintf(' & ');
fprintf('%5.5g',inf(r.q_max));
fprintf(' & ');
fprintf('%3.3g',sup(r.M_x_n1));
fprintf(' & ');
fprintf('%3.3g',sup(r.M_q_n1));
fprintf(' & ');
fprintf('%3.3g',sup(r.rho_q));
fprintf(' & ');
fprintf('%g',r.N_x_n1);
fprintf(' & ');
fprintf('%g',r.N_q_n1);
fprintf(' & ');
fprintf('%3.3g',d.MN(j));
fprintf('\\\\\\\\');

if sup(r.err_x_n1) > 1e-17
    error('interpolation error greater than claimed')
end

if sup(r.err_q_n1) > 1e-17
    error('interpolation error greater than claimed')
end

end

fprintf('\n\\hline\n\\endtabular\n\\captionTODO  %%Table created by create_tableA.m\n\\labeltable:lower-instability\n\\endtable');

fprintf('\n\n\n');

```

3.16 distinct.m

```
clear all; curr_dir = local_startup;

pie = nm('pi');

Kvals = [
    nm('0.999999'), nm('0.9999995');
    nm('0.99999'), nm('0.999999');
    nm('0.9999'), nm('0.99999');
    nm('0.999'), nm('0.9999');
    nm('0.99'), nm('0.999');
    nm('0.9'), nm('0.99');
];

N = size(Kvals,1);
n = 10;

for ind = 1:N

    left = Kvals(ind,1); % left side of interval in k
    right = Kvals(ind,2); % right side of interval in k
    del = (right-left)/n; % width of sub interval in k

    for j = 1:n

        clc;
        fprintf('Verifying alpha_j distinct: Percent done = %4.4g\n',100*((ind-
1)*n+j)/(N*n));
        % interval in k
        k = left + nm((j-1)*del,j*del);

        success = check_alpha_distinct(k);
        if success == 0
            error('Failed to verify the alpha_j are distinct.');
```

```
end
```

3.17 divide_interpolation.m

```
clear all; curr_dir = local_startup;
```

Finds the 2d Chebyshev polynomials with $q \in [q_{\min}, q_{\max}]$.

```
cnt = 0;
```

```
cnt = cnt + 1;  
q_mincnt = nm('0.65'); % q_min  
q_maxcnt = nm('0.71'); % q_max  
% these next two numbers break up the domain for  
% finding a lower bound on the theta function that  
% shows up in the definition of v(x)  
num_stepscnt = 5000 ;  
stepscnt = 20000 ;  
% name by which this will be saved  
file_namecnt = 'interp2d_650_to_710';
```

```
cnt = cnt + 1;  
q_mincnt = nm('0.59');  
q_maxcnt = nm('0.66');  
num_stepscnt = 12000 ;  
stepscnt = 10000 ;  
file_namecnt = 'interp2d_590_to_660';
```

```
cnt = cnt + 1;  
q_mincnt = nm('0.53');
```

```

q_maxcnt = nm('0.6');
num_stepscnt = 12000 ;
stepscnt = 10000 ;
file_namecnt = 'interp2d_530_to_600';

cnt = cnt + 1;
q_mincnt = nm('0.49');
q_maxcnt = nm('0.538');
num_stepscnt = 600 ;
stepscnt = 8000 ;
file_namecnt = 'interp2d_490_to_538';

cnt = cnt + 1;
q_mincnt = nm('0.35');
q_maxcnt = nm('0.5');
num_stepscnt = 600 ;
stepscnt = 8000 ;
file_namecnt = 'interp2d_350_to_500';

cnt = cnt + 1;
q_mincnt = nm('0.1');
q_maxcnt = nm('0.4');
num_stepscnt = 100 ;
stepscnt = 8000 ;
file_namecnt = 'interp2d_100_to_400';

parfor j = 1:length(q_min)

    interpolation_2d(q_minj,q_maxj,num_stepsj,stepsj,file_namej);

end

```

3.18 driver.m

```
%  
%  
% alpha_j distinct  
%
```

Verify that the α_j are distinct for $k \in [0.9, 0.9999995]$. Distinctness of the α_j is required for showing stability of the periodic waves.

```
distinct;
```

```
%  
%  
% simplicity of eigenvalues  
%
```

```
simplicity;
```

```
%  
%  
% lower instability  
%
```

Verify spectral instability for a region below the stability region by interpolating in the variable q with the variable $npsi$ fixed at 1.

```
lower_instability_interpolation;
```

```
%  
%  
% 2d interpolation  
%
```

```

divide_interpolation;

%-----
% middle stability region
%-----

% verify for  $k \in [0.93, 0.9999983]$  that the case
%  $\alpha = i\psi\omega$  is consistent with stability

%  $\alpha = i\psi\omega$ 
driver_stability_n0;

%  $\alpha = \omega + i\psi\omega$ 
driver_stability_n1;

%-----
% middle stability region
%-----

driver_instability_upper;

%-----
% strict upper
%-----

driver_strict_upper_taylor;

driver_pinpoint;

driver_strict_lower

%-----
% aux lemma
%-----

```

```
kappa_lemma;
```

```
lemma_qk;
```

3.19 driver_instability_upper.m

```
clear all; curr_dir = local_startup;
```

```
psi_left = 0.6;  
psi_right = 0.8;
```

```
ints_x = 1;  
ints_y = 1000;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
file_name = 'interp2d_490_to_538';  
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');  
d = ld.var;  
cnt = 0;
```



```

cnt = cnt + 1;
stcnt1 = nm('0.99999839');
stcnt2 = nm('0.9999984');
stcnt3 = 200; % number of k intervals

```

```

cnt = cnt + 1;
stcnt1 = nm('0.9999984');
stcnt2 = nm('0.9999985');
stcnt3 = 500; % number of k intervals

```

```

cnt = cnt + 1;
stcnt1 = nm('0.9999985');
stcnt2 = nm('0.9999986');
stcnt3 = 100; % number of k intervals

```

```

cnt = cnt + 1;
stcnt1 = nm('0.9999986');
stcnt2 = nm('0.9999987');
stcnt3 = 50; % number of k intervals

```

```

cnt = cnt + 1;
stcnt1 = nm('0.9999987');
stcnt2 = nm('0.9999988');
stcnt3 = 50; % number of k intervals

```

```

cnt = cnt + 1;
stcnt1 = nm('0.9999988');
stcnt2 = nm('0.9999989');
stcnt3 = 50; % number of k intervals

```

```

cnt = cnt + 1;
stcnt1 = nm('0.9999989');
stcnt2 = nm('0.999999');
stcnt3 = 50; % number of k intervals

```

```

for ind = 1:cnt
    kvals = linspace(inf(stind1),sup(stind2),stind3+1);
    for j = length(kvals)-1
        clc;
    end
end

```

```

        fprintf('upper instability, k: %16.16g\n',mid(kvals(j)));
        verify_instability_upper(d, kvals(j), kvals(j+1),psi_left,psi_right,ints_y);
    end
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

file_name = 'interp2d_530_to_600';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
cnt = 0;

```

```

cnt = cnt + 1;
stcnt1 = nm('0.999999');
stcnt2 = nm('0.9999999');
stcnt3 = 500; % number of k intervals

```

```

cnt = cnt + 1;
stcnt1 = nm('0.99999999');
stcnt2 = nm('0.99999995');
stcnt3 = 500; % number of k intervals

```

```

for ind = 1:cnt
    kvals = linspace(inf(stind1),sup(stind2),stind3+1);
    for j = length(kvals)-1
        clc;
        fprintf('upper instability, k: %16.16g\n',mid(kvals(j)));
        verify_instability_upper(d, kvals(j), kvals(j+1),psi_left,psi_right,ints_y);
    end
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

file_name = 'interp2d_590_to_660';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
cnt = 0;

```

```

cnt = cnt + 1;
stcnt1 = nm('0.99999995');
stcnt2 = nm('0.99999999');
stcnt3 = 500; % number of k intervals

cnt = cnt + 1;
stcnt1 = nm('0.99999999');
stcnt2 = nm('0.9999999993');
stcnt3 = 500; % number of k intervals

for ind = 1:cnt
    kvals = linspace(inf(stind1),sup(stind2),stind3+1);
    for j = length(kvals)-1
        clc;
        fprintf('upper instability, k: %16.16g\n',mid(kvals(j)));
        verify_instability_upper(d, kvals(j), kvals(j+1),psi_left,psi_right,ints_y);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

file_name = 'interp2d.650_to.710';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
cnt = 0;

cnt = cnt + 1;
stcnt1 = nm('0.9999999993');
stcnt2 = nm('0.9999999999');
stcnt3 = 500; % number of k intervals

cnt = cnt + 1;
stcnt1 = nm('0.9999999999');
stcnt2 = nm('0.99999999999');
stcnt3 = 500; % number of k intervals

cnt = cnt + 1;
stcnt1 = nm('0.999999999999');
stcnt2 = nm('0.999999999997');

```

```
stcnt3 = 500; % number of k intervals
```

```
for ind = 1:cnt
    kvals = linspace(inf(stind1),sup(stind2),stind3+1);
    for j = length(kvals)-1
        clc;
        fprintf('upper instability, k: %16.16g\n',mid(kvals(j)));
        verify_instability_upper(d, kvals(j), kvals(j+1),psi_left,psi_right,ints_y);
    end
end
```

3.20 driver_stability_n0.m

```
% clear all; curr_dir = local_startup;
%
% % _____
% % k = 0.93 - 0.9997
% % _____
```

```
file_name = 'interp2d_100_to_400';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
```

```
ints_yL = 1200;
ints_yR = 1000;
psi_L = nm('0.001');
psi_M = nm('0.5');
psi_R = nm('0.7');
psi_R2 = nm('0.8');
psi_R3 = nm('0.9');
stats = 'off';
pnts = 1500;
cnt = 0;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.93'); % left k value
```

```

stcnt2 = nm('0.99'); % right k value
stcnt3 = 800; % kvals

cnt = cnt + 1;
stcnt1 = nm('0.99');
stcnt2 = nm('0.999');
stcnt3 = 100;

cnt = cnt + 1;
stcnt1 = nm('0.999');
stcnt2 = nm('0.9997');
stcnt3 = 100;

for ind = 1:cnt
    kvals = (linspace(inf(stcnt1),sup(stcnt2),stcnt3));
    parfor j = 1:length(kvals)-1

        local_startup_batch(curr_dir);
        clc;
        j

        verify_stability_n0_strict(d, kvals(j), kvals(j+1),ints_yL,...
            ints_yR,psi_L,psi_M,psi_R,psi_R2,psi_R3,stats);

        verify_stability_single(d,nm(kvals(j),kvals(j+1)),pnts,psi_R,psi_R2)

    end
end

% -----
% k = 0.9997 - 0.99999
% -----

clc;
file_name = 'interp2d_350_to_500';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
cnt = 0;

```

```

cnt = cnt + 1;
stcnt1 = nm('0.9997');
stcnt2 = nm('0.9999');
stcnt3 = 600;

cnt = cnt + 1;
stcnt1 = nm('0.9999');
stcnt2 = nm('0.999993');
stcnt3 = 600;

for ind = 1:cnt
    kvals = fliplr(linspace(inf(stcnt1),sup(stcnt2),stcnt3));
    parfor j = 1:length(kvals)-1

        local_startup_batch(curr_dir);
        clc;

        verify_stability_n0_strict(d, kvals(j), kvals(j+1),ints_yL,...
            ints_yR,psi_L,psi_M,psi_R,psi_R2,psi_R3,stats);

        verify_stability_single(d,nm(kvals(j),kvals(j+1)),pnts,psi_R,psi_R2)

    end
end

% -----
% k = 0.99999 - 0.9999983
% -----

clc;
file_name = 'interp2d_490_to_538';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
cnt = 0;

```

```
cnt = cnt + 1;
stcnt1 = nm('0.999993');
stcnt2 = nm('0.999997');
stcnt3 = 200;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.999997');
stcnt2 = nm('0.9999978');
stcnt3 = 100;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.9999977');
stcnt2 = nm('0.9999978');
stcnt3 = 10;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.9999978');
stcnt2 = nm('0.9999979');
stcnt3 = 10;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.9999979');
stcnt2 = nm('0.999998');
stcnt3 = 20;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.999998');
stcnt2 = nm('0.9999981');
stcnt3 = 20;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.9999981');
stcnt2 = nm('0.9999982');
stcnt3 = 30;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.9999982');
stcnt2 = nm('0.9999983');
```

```

stcnt3 = 100;

for ind = 1:cnt
    kvals = fliplr(linspace(inf(stcnt1),sup(stcnt2),stcnt3));
    parfor j = 1:length(kvals)-1

        local_startup_batch(curr_dir);
        clc;

        verify_stability_n0_strict(d, kvals(j), kvals(j+1),ints_yL,...
            ints_yR,psi_L,psi_M,psi_R,psi_R2,psi_R3,stats);

        verify_stability_single(d,nm(kvals(j),kvals(j+1)),pnts,psi_R,psi_R2)

    end
end

```

3.21 driver_stability_n1.m

```

clear all; curr_dir = local_startup;

% keepf fixed
ints_x = 1;

% _____
% k = 0.93 - 0.9997
% _____

clc;
file_name = 'interp2d_100_to_400';

```



```
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
cnt = 0;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.9426');
stcnt2 = nm('0.943');
stcnt3 = 11;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.943');
stcnt2 = nm('0.944');
stcnt3 = 11;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.944');
stcnt2 = nm('0.945');
stcnt3 = 11;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.945');
stcnt2 = nm('0.946');
stcnt3 = 11;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.946');
stcnt2 = nm('0.947');
stcnt3 = 11;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.947');
stcnt2 = nm('0.948');
stcnt3 = 11;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.948');
stcnt2 = nm('0.949');
stcnt3 = 11;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.949');
stcnt2 = nm('0.95');
stcnt3 = 20;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.95');
stcnt2 = nm('0.96');
stcnt3 = 11;
stcnt4 = 3000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.96');
stcnt2 = nm('0.97');
stcnt3 = 11;
stcnt4 = 1000;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.97');
stcnt2 = nm('0.999');
stcnt3 = 200;
stcnt4 = 200;
```

```
cnt = cnt + 1;
stcnt1 = nm('0.999');
stcnt2 = nm('0.9997');
stcnt3 = 20;
stcnt4 = 200;
```

```
for ind = 1:cnt
```

```

kvals = linspace(inf(stind1),sup(stind2),stind3+1);
ints_y = stind4;
for j = 1:length(kvals)-1

    clc;
    fprintf('k = %16.16g\n',mid(kvals(j)));

    verify_stability_n1(d, kvals(j), kvals(j+1),ints_x,ints_y);

end
end

% -----
% k = 0.9997 - 0.99999
% -----

clc;
file_name = 'interp2d_350_to_500';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
cnt = 0;

cnt = cnt + 1;
stent1 = nm('0.9997');
stent2 = nm('0.9999');
stent3 = 11;
stent4 = 3000;

cnt = cnt + 1;
stent1 = nm('0.9999');
stent2 = nm('0.999993');
stent3 = 100;
stent4 = 3000;

for ind = 1:cnt

```

```

kvals = linspace(inf(stind1),sup(stind2),stind3+1);
ints_y = stind4;
for j = 1:length(kvals)-1

    clc;
    fprintf('k = %16.16g\n',mid(kvals(j)));

    verify_stability_n1(d, kvals(j), kvals(j+1),ints_x,ints_y);

end
end

% -----
% k = 0.99999 - 0.9999983
% -----

clc;
file_name = 'interp2d_490_to_538';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
cnt = 0;

cnt = cnt + 1;
stent1 = nm('0.999993');
stent2 = nm('0.999994');
stent3 = 20;
stent4 = 3000;

cnt = cnt + 1;
stent1 = nm('0.999994');
stent2 = nm('0.999995');
stent3 = 20;
stent4 = 3000;

cnt = cnt + 1;
stent1 = nm('0.999995');

```

```

stcnt2 = nm('0.999996');
stcnt3 = 20;
stcnt4 = 3000;

cnt = cnt + 1;
stcnt1 = nm('0.999996');
stcnt2 = nm('0.999997');
stcnt3 = 20;
stcnt4 = 3000;

cnt = cnt + 1;
stcnt1 = nm('0.999997');
stcnt2 = nm('0.999998');
stcnt3 = 20;
stcnt4 = 3000;

cnt = cnt + 1;
stcnt1 = nm('0.999998');
stcnt2 = nm('0.999999');
stcnt3 = 100;
stcnt4 = 3000;

for ind = 1:cnt

    kvals = linspace(inf(stind1),sup(stind2),stind3+1);
    ints_y = stind4;
    for j = 1:length(kvals)-1

        clc;
        fprintf('k = %16.16g\n',mid(kvals(j)));

        verify_stability_n1(d, kvals(j), kvals(j+1),ints_x,ints_y);

    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

3.22 driver_strict_lower.m

```
clear all; curr_dir = local_startup;
intvalinit('DisplayMidRad');
% return

file_name = 'd_stability10';
ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data');
d = ld.var;

pie = nm('pi');

%
% minimum distance between the two stadiums
%

rho_q_sm = ((d.rho_q - 1/d.rho_q)+sqrt((d.rho_q-1/d.rho_q)^2+16))/4;
rho_psi_sm = ((d.rho_psi - 1/d.rho_psi)+sqrt((d.rho_psi-1/d.rho_psi)^2+16))/4;
rho_q = d.rho_q;
rho_psi = d.rho_psi;

nm_pts = 100;
min_diff_q = inf((rho_q-1/rho_q)/2);
for j = 0:nm_pts-1
    theta1 = nm(2*(j*pie)/nm_pts,2*((j+1)*pie)/nm_pts);
    for q = 0:nm_pts-1
        theta2 = nm(2*(q*pie)/nm_pts,2*((q+1)*pie)/nm_pts);
        del = (exp(1i*theta1)*rho_q+exp(-1i*theta1)/rho_q ...
            -(exp(1i*theta2)*rho_q_sm+exp(-1i*theta2)/rho_q_sm))/2;
```

```

        min_diff_q = min(min_diff_q,inf(abs(del)));
        if min_diff_q == 0
            error('min is 0')
        end
    end
end
min_diff_q = min_diff_q/2;
% min_diff_q = 0.150639673896165;

kvals = [nm('0.942197747747748','0.9422'), ...
        nm('0.9422','0.9423'), nm('0.9423','0.9424'), ...
        nm('0.9424','0.9425'), nm('0.9425','0.9426')];

psi0 = nm('1');
psiL = nm('0.99');

tic
for j = 1:length(kvals)

    fprintf('\n\n\n');
    strict_transition_lower(d,kvals(j),psi0,psiL,min_diff_q,rho_q_sm)

end
toc

disp('verify stability')

ints_x = 1;
ints_y = 8000;
divisions = 2;

for k = 1:length(kvals)
    kv = linspace(inf(kvals(k)),sup(kvals(k)),divisions);
    for j = 1:length(kv)-1

        kleft = kv(j);
        kright = kv(j+1);

```

```

        verify_stability_n1_strict(d, kleft, kright, ints_x,ints_y,psiL)

    end
end

```

3.23 driver_strict_upper_taylor.m

```

clear all; curr_dir = local_startup;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

file_name = 'interp2d_490_to_538';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-----
% k and psi points/intervals
%-----

num_k = 1001;
kv = linspace(inf(nm('0.9999983')),sup(nm('0.99999839')),num_k);
kv = nm(kv(1:end-1),kv(2:end));

psiv = nm('0.7','0.8');
num = 100;

psi0 = linspace(inf(psiv(1)),sup(psiv(end)),num);
psi0 = nm(psi0(1:end-1),psi0(2:end));

```



```

%-----
% retrieve file
%-----

ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;
pie = nm('pi');

%-----
% diff
%-----

rho_q_sm = ((d.rho_q - 1/d.rho_q)+sqrt((d.rho_q-1/d.rho_q)^2+16))/4;
rho_psi_sm = ((d.rho_psi - 1/d.rho_psi)+sqrt((d.rho_psi-1/d.rho_psi)^2+16))/4;
rho_q = d.rho_q;
rho_psi = d.rho_psi;

nm_pts = 100;
min_diff_q = inf((rho_q-1/rho_q)/2);
for j = 0:nm_pts-1
    theta1 = nm(2*(j*pie)/nm_pts,2*((j+1)*pie)/nm_pts);
    for q = 0:nm_pts-1
        theta2 = nm(2*(q*pie)/nm_pts,2*((q+1)*pie)/nm_pts);
        del = (exp(1i*theta1)*rho_q+exp(-1i*theta1)/rho_q ...
            -(exp(1i*theta2)*rho_q_sm+exp(-1i*theta2)/rho_q_sm))/2;
        min_diff_q = min(min_diff_q,inf(abs(del)));
        if min_diff_q == 0
            error('min is 0')
        end
    end
end
min_diff_q = min_diff_q/2;

nm_pts = 100;
min_diff_psi = inf((rho_psi-1/rho_psi)/2);
for j = 0:nm_pts-1
    theta1 = nm(2*(j*pie)/nm_pts,2*((j+1)*pie)/nm_pts);
    for q = 0:nm_pts-1
        theta2 = nm(2*(q*pie)/nm_pts,2*((q+1)*pie)/nm_pts);
        del = (exp(1i*theta1)*rho_psi+exp(-1i*theta1)/rho_psi ...
            -(exp(1i*theta2)*rho_psi_sm+exp(-1i*theta2)/rho_psi_sm))/2;

```

```

        min_diff_psi = min(min_diff_psi,inf(abs(del)));
        if min_diff_psi == 0
            error('min is 0')
        end
    end
end
min_diff_psi = min_diff_psi/2;

for j = 1:length(kv)

    fprintf('\npercent done: %4.4g',100*j/num_k);
    k = kv(j);

    [mx_fk,mn_fpsipsi] = strict_transition(k,psi0,min_diff_q,min_diff_psi,d);

    if mx_fk >= 0
        error('failed to verify strict transition on the region')
    end
    if mn_fpsipsi <= 0
        error('failed to verify strict transition on the region')
    end
end
end

```

3.24 elliptic_integral.m

```

function out = elliptic_integral(k,id)
% function out = elliptic_integral(k,id)
%
% Compute elliptic integrals using interval arithmetic.
%
% input k should be an intlab interval. The input id
% denotes the type of elliptic integral.
%
% id = 1, complete elliptic integral of the first kind
% id = 2, complete elliptic integral of the second kind
%
% The complete elliptic integral of the first kind is solved using
% the arithmetic geometric mean. The complete elliptic integral
% of the second kind is solved using Carlson's algorithm.

% complete elliptic integral of the first kind
%
%  $\int_0^1 \frac{1}{\sqrt{1-x^2}\sqrt{1-k^2x^2}} dx$ 

M = agm(ones(size(k)),sqrt(1-k.^2));
out = nm('pi')./(2*M);

if id == 1
    return;
end

%
% complete elliptic integral of the second kind
%
%  $\int_0^{2\pi} \sqrt{1-k^2 \sin^2(\theta)} d\theta$ 

% compute m = 0 case
x = nm(zeros(size(k)));
y = 1-k.^2;
z = nm(ones(size(k)));
lambda = sqrt(x.*y)+sqrt(x.*z)+sqrt(y.*z);
% mu = (x+y+3*z)/5;

```

```

sum = 1./sqrt(z).*(z+lambda));
% X = 1-x./mu;
% Y = 1-y./mu;
% Z = 1-z./mu;

% t1 = hull(nm(zeros(size(k))),abs(X));
% t2 = hull(abs(Y),abs(Z));
% eps = hull(t1,t2);

% r = abs(3*eps.^6./((1-eps).^(3/nm(2))));
%
% S2 = (X.^2+Y.^2+3*Z.^2)/4;
% S3 = (X.^3+Y.^3+3*Z.^3)/6;
% S4 = (X.^4+Y.^4+3*Z.^4)/8;
% S5 = (X.^5+Y.^5+3*Z.^5)/10;

% compute m = 1 case;
m = 1;

x = (x+lambda)/4;
y = (y+lambda)/4;
z = (z+lambda)/4;
lambda = sqrt(x.*y)+sqrt(x.*z)+sqrt(y.*z);
mu = (x+y+3*z)/5;
X = 1-x./mu;
Y = 1-y./mu;
Z = 1-z./mu;

t1 = hull(nm(zeros(size(k))),abs(X));
t2 = hull(abs(Y),abs(Z));
eps = hull(t1,t2);
r = abs(3*eps.^6./((1-eps).^(3/nm(2))));

S2 = (X.^2+Y.^2+3*Z.^2)/4;
S3 = (X.^3+Y.^3+3*Z.^3)/6;
S4 = (X.^4+Y.^4+3*Z.^4)/8;
S5 = (X.^5+Y.^5+3*Z.^5)/10;

factor = 4^nm(-m)*mu.^(-nm(3)/2).*(1+...
    (nm(3)/7)*S2+(nm(1)/3)*S3+...

```

```

(nm(3)/22)*S2.^2 + (nm(3)/11)*S4+...
(nm(3)/13)*S2.*S3+(nm(3)/13)*S5 + r);

Rd_new = 3*sum + factor;

sum = sum + 4^(-nm(m))./(sqrt(z).*(z+lambda));

Rd_old = infsup(-1+inf(Rd_new),1+sup(Rd_new));

m = 2;
while (max(sup(Rd_old))-min(inf(Rd_old)))- (max(sup(Rd_new))-min(inf(Rd_new)))-
> 0

Rd_old = Rd_new;

x = (x+lambda)/4;
y = (y+lambda)/4;
z = (z+lambda)/4;
lambda = sqrt(x.*y)+sqrt(x.*z)+sqrt(y.*z);
mu = (x+y+3*z)./5;
X = 1-x./mu;
Y = 1-y./mu;
Z = 1-z./mu;

t1 = hull(nm(zeros(size(k))),abs(X));
t2 = hull(abs(Y),abs(Z));
eps = hull(t1,t2);
r = abs(3*eps.^6./(1-eps).^(nm(3)/2));

S2 = (X.^2+Y.^2+3*Z.^2)/4;
S3 = (X.^3+Y.^3+3*Z.^3)/6;
S4 = (X.^4+Y.^4+3*Z.^4)/8;
S5 = (X.^5+Y.^5+3*Z.^5)/10;

factor = 4^nm(-m)*mu.^(-nm(3)/2).*(1+...
(nm(3)/7)*S2+(nm(1)/3)*S3+...
(nm(3)/22)*S2.^2 + (nm(3)/11)*S4+...
(nm(3)/13)*S2.*S3+(nm(3)/13)*S5 + r);

```

```

Rd_new = 3*sum + factor;

sum = sum + 4^(-nm(m))./(sqrt(z).*(z+lambda));

m = m + 1;

end

if id == 2
    % complete elliptic integral of the second kind.
    out = out -(k.^2/3).*Rd_new;
elseif id == 3
    % The difference of elliptic integrals, K-E
    out = (k.^2/3).*Rd_new;
end

```

3.25 instability_interpolation.m

```

function d = instability_interpolation(d,q_min,q_max)

pie = nm('pi');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
d.a_q = q_min;
d.b_q = q_max;

```

Note that ρ_ψ must be chosen so that $\xi(\omega + i\psi\omega')$ does not have a pole. The poles of ξ are $z = 2m\omega + 2n\omega'$. Setting $2m\omega + 2n\omega' = \omega + i\psi\omega'$ with $\psi = 1/2 + \tilde{\psi}/2$, we find that $|\Im(\tilde{\psi})| < -\frac{\pi}{\log(q)}$ is necessary and sufficient to ensure analyticity.

```
%%
```

```

% find rho_psi
c_psi = nm('0.9')*pie/abs(log(q_min));

```

```

rho_psi = nm(inf(c_psi + sqrt(c_psi^2+1)));
d.rho_psi = rho_psi;

% make sure rho_psi is small enough that bounds
% on xi(omega + 1i*psi*omega') are valid
if sup(rho_psi) >= 3+sqrt(nm(2))
    rho_psi = 3+sqrt(nm(2));
end

if sup(rho_psi) <= 1
    error('problem');
end

%
% find rho_q
%

a = q_min;
b = q_max;
d.q_min = q_min;
d.q_max = q_max;

rho1 = (b+a)/(b-a)-2*sqrt(a*b)/(b-a);
rho2 = (b+a)/(b-a)+2*sqrt(a*b)/(b-a);
rho3 = (2-a-b)/(b-a)-2*sqrt(1-a-b+a*b)/(b-a);
rho4 = (2-a-b)/(b-a)+2*sqrt(1-a-b+a*b)/(b-a);

rho_left = nm(rho1,rho3);
rho_right = nm(rho2,rho4);
if sup(rho_left) >= inf(rho_right)
    error('problem');
end

rho_left = nm(sup(rho_left));
rho_right = nm(inf(rho_right));
rho_q = rho_left + nm('0.9')*(rho_right-rho_left);
d.rho_q = rho_q;

%

```

```

% get bounds for  $\alpha = \omega + i\psi\omega$ 
%

ntilde = 1;

% bound_numer(dm,rho_psi,rho_x,rho_q,q_min,q_max,nm(0),nm(1),ntilde);
[M_psi,M_x,M_q] = bound_numer_unstable(d.dm,d.rho_x,d.rho_q,q_min,q_max);

if isnan(sup(M_psi))
    error('Nan for bound')
end
if isnan(sup(M_x))
    error('Nan for bound')
end
if isnan(sup(M_q))
    error('Nan for bound')
end
if isinf(sup(M_psi))
    error('Infinity for bound')
end
if isinf(sup(M_x))
    error('Infinity for bound')
end
if isinf(sup(M_q))
    error('Infinity for bound')
end

abs_tol = 1e-17;
[N_x,err_x] = N_nodes(d.rho_x,M_x,abs_tol);
d.N_x = N_x;
d.err_x = err_x;

[N_q,err_q] = N_nodes(rho_q,M_q,abs_tol);

%-----
% get interpolation coefficients when ntilde = 1
%-----

a_q = a;

```



```

b_q = b;

fun = (q,psi)(integrand_numer(N_x,err_x,q*(b_q-a_q)/2+(a_q+b_q)/2,1,ntilde));

% interpolation coefficients
t1 = tic;

N_psi = 0; % KEEP! this is sufficient for instability computation

cfn1 = cf_biv_cheby(N_q,N_psi,6,fun);
d.cfl_time = toc(t1);
d.cfn1 = cfn1;

d.M_psi_n1 = M_psi;
d.M_q_n1 = M_q;
d.M_x_n1 = M_x;
d.N_psi_n1 = N_psi;
d.N_q_n1 = N_q;
d.N_x_n1 = N_x;
d.err_q_n1 = err_q;
d.err_x_n1 = err_x;
d.a_q = a_q;
d.b_q = b_q;
d.cfn1 = cfn1;
d.rho_q_n1 = rho_q;

```

3.26 integrand_numer.m

```

function out = integrand_numer(Nx,err,q,psi,ntilde)

xi = xi_q_psi(q,psi,ntilde);
xi_der = xi_der_q_psi(q,psi,ntilde);

pie = nm('pi');

```

```

half = nm(1)/2;
theta = ((0:1:Nx)+half)*pie/(Nx+1);
x = cos(theta);

% alpha = 0
J = theta_vec(q,0,x,3,0);
J = repmat(J,[1 length(psi)]);

E0 = 1./J(:,1);
E1 = -J(:,2)./J(:,1).^2;
E2 = 2*J(:,2).^2./J(:,1).^3-J(:,3)./J(:,1).^2;
E3 = -6*J(:,2).^3./J(:,1).^4+6*J(:,2).*J(:,3)./J(:,1).^3-J(:,4)./J(:,1).^2;

B0 = E0.*E0;
B1 = 2*E0.*E1;
B2 = 2*(E0.*E2+E1.*E1);
B3 = 2*(E3.*E0+3*E1.*E2);

% alpha \neq 0
L = theta_vec(q,psi,x,4,ntilde);

A0 = L(:,1).*L(:,1);
A1 = 2*L(:,1).*L(:,2);
A2 = 2*(L(:,1).*L(:,3)+L(:,2).*L(:,2));
A3 = 2*(L(:,4).*L(:,1)+3*L(:,2).*L(:,3));
A4 = 2*(L(:,1).*L(:,5)+4*L(:,2).*L(:,4)+3*L(:,3).*L(:,3));

w0 = A0.*B0;
w1 = A0.*B1+A1.*B0;
w2 = A0.*B2+2*A1.*B1+A2.*B0;
w3 = A0.*B3+3*A1.*B2+3*A2.*B1+ A3.*B0;

con = log(q)/(1i*pie);
D0 = con*(A1.*B0);
D1 = con*(A1.*B1+A2.*B0);
D2 = con*(A1.*B2+2*A2.*B1+A3.*B0);
D3 = con*(A1.*B3+3*A2.*B2+3*A3.*B1+ A4.*B0);

```

```

xicon = li*repmat(xi,length(x),1);
xicon_der = li*repmat(xi_der,length(x),1);

xicon2 = xicon.*xicon;
xicon3 = xicon.*xicon2;

v1 = w1+xicon.*w0;
v2 = w2+2*xicon.*w1+xicon2.*w0;
v3 = w3+3*xicon.*w2+3*xicon2.*w1+xicon3.*w0;

v1_psi = D1 + xicon_der.*w0+xicon.*D0;

v2_psi = D2+2*xicon_der.*w1+2*xicon.*D1+...
        2*xicon.*xicon_der.*w0+xicon2.*D0;

v3_psi = D3 + 3*xicon_der.*w2+3*xicon.*D2+6*xicon.*xicon_der.*w1...
        +3*xicon2.*D1+3*xicon2.*xicon_der.*w0+xicon3.*D0;

f1 = v1.*conj(v2);
f1_psi = v1_psi.*conj(v2)+v1.*conj(v2_psi);

f2 = v3.*conj(v2);
f2_psi = v3_psi.*conj(v2)+v3.*conj(v2_psi);

g = w0.*conj(v1);
g_psi = D0.*conj(v1)+w0.*conj(v1_psi);

% Chebyshev polynomials evaluated at the points
Tx = cos(theta.*(0:2:Nx));

cf1 = 2*f1.'*Tx/(Nx+1);
cf1(:,1) = cf1(:,1)/2;
cf2 = 2*f2.'*Tx/(Nx+1);
cf2(:,1) = cf2(:,1)/2;

```

```

cg = 2*g.*Tx/(Nx+1);
cg(:,1) = cg(:,1)/2;

cf1_psi = 2*f1_psi.*Tx/(Nx+1);
cf1_psi(:,1) = cf1_psi(:,1)/2;
cf2_psi = 2*f2_psi.*Tx/(Nx+1);
cf2_psi(:,1) = cf2_psi(:,1)/2;
cg_psi = 2*g_psi.*Tx/(Nx+1);
cg_psi(:,1) = cg_psi(:,1)/2;

out = nm(zeros(length(psi),1,6));

out(:,1) = 2*cf1*(1./(1-(0:2:Nx).^2)).';
out(:,3) = 2*cf2*(1./(1-(0:2:Nx).^2)).';
out(:,5) = 2*cg*(1./(1-(0:2:Nx).^2)).';

out(:,2) = 2*cf1_psi*(1./(1-(0:2:Nx).^2)).';
out(:,4) = 2*cf2_psi*(1./(1-(0:2:Nx).^2)).';
out(:,6) = 2*cg_psi*(1./(1-(0:2:Nx).^2)).';

% add integration ( in x ) error
out = out + 2*(nm(-err,err)+1i*nm(-err,err));

```

3.27 interpolation_2d.m

```
function interpolation_2d(q_min,q_max,num_steps,steps,file_name)

curr_dir = cd;
local_startup_batch(cd);

total_time = tic;

%
% Get the lower bound of theta
%

pie = nm('pi');
frac = nm('0.9');

t1 = tic;
[dm,rho_x] = lower_bound(frac,q_min,q_max,num_steps,steps);
d.dm_time = toc(t1); %

d.dm = dm;
d.rho_x = rho_x;

Note that  $\rho_\psi$  must be chosen so that  $\xi(\omega + i\psi\omega')$  does not have a pole. The poles
of  $\xi$  are  $z = 2m\omega + 2n\omega'$ . Setting  $2m\omega + 2n\omega' = \omega + i\psi\omega'$  with  $\psi = 1/2 + \tilde{\psi}/2$ ,
we find that  $|\Im(\tilde{\psi})| < -\frac{\pi}{\log(q)}$  is necessary and sufficient to ensure analyticity.

%%

% find rho_psi
c_psi = nm('0.9')*pie/abs(log(q_min));
rho_psi = nm(inf(c_psi + sqrt(c_psi^2+1)));
d.rho_psi = rho_psi;

% make sure rho_psi is small enough that bounds
```

```

% on  $\text{xi}(\omega + \text{li} \cdot \text{psi} \cdot \omega)$  are valid
if sup(rho_psi) >= 3+sqrt(nm(2))
    rho_psi = 3+sqrt(nm(2));
end

if sup(rho_psi) <= 1
    error('problem');
end

%
% find rho_q
%

a = q_min;
b = q_max;
d.q_min = q_min;
d.q_max = q_max;

rho1 = (b+a)/(b-a)-2*sqrt(a*b)/(b-a);
rho2 = (b+a)/(b-a)+2*sqrt(a*b)/(b-a);
rho3 = (2-a-b)/(b-a)-2*sqrt(1-a-b+a*b)/(b-a);
rho4 = (2-a-b)/(b-a)+2*sqrt(1-a-b+a*b)/(b-a);

rho_left = nm(rho1,rho3);
rho_right = nm(rho2,rho4);
if sup(rho_left) >= inf(rho_right)
    error('problem');
end

rho_left = nm(sup(rho_left));
rho_right = nm(inf(rho_right));
rho_q = rho_left + nm('0.9')*(rho_right-rho_left);
d.rho_q = rho_q;

%
% get bounds for  $\alpha = \omega + \text{li} \cdot \text{psi} \cdot \omega$ 
%

% ntilde = 1

```

```

ntilde = 1;

time1 = tic;

[M_psi,M_x,M_q] = bound_numer(dm,rho_psi,rho_x,rho_q,q_min,q_max,nm(0),nm(1),ntilde);
toc(time1)

abs_tol = 1e-17;
[N_x,err_x] = N_nodes(rho_x,M_x,abs_tol);

abs_tol = 1e-17;
[N_psi,err_psi] = N_nodes(rho_psi,M_psi,abs_tol);

[N_q,err_q] = N_nodes(rho_q,M_q,abs_tol);

d.bd_n1_time = toc(time1);

%-----
% get interpolation coefficients when ntilde = 1
%-----

a_q = a;
b_q = b;
a_psi = nm(0);
b_psi = nm(1);

fun = (q,psi)(integrand_numer(N_x,err_x,q*(b_q-a_q)/2+(a_q+b_q)/2,...
    psi*(b_psi-a_psi)/2+(a_psi+b_psi)/2,ntilde));

% interpolation coefficients
t1 = tic;
cfn1 = cf_biv_cheby(N_q,N_psi,6,fun);

```

```

d.cfl_time = toc(t1);
d.cfn1 = cfn1;

d.M_psi_n1 = M_psi;
d.M_q_n1 = M_q;
d.M_x_n1 = M_x;
d.N_psi_n1 = N_psi;
d.N_q_n1 = N_q;
d.N_x_n1 = N_x;
d.err_psi_n1 = err_psi;
d.err_q_n1 = err_q;
d.err_x_n1 = err_x;
d.a_q = a_q;
d.b_q = b_q;
d.cfn1 = cfn1;
d.dm = dm;
d.rho_psi_n1 = rho_psi;
d.rho_q_n1 = rho_q;
d.rho_x_n1 = rho_x;

%
% get bounds on 10 integrands for alpha = 1i*psi*omega'
%

[M_psi_10,M_x_10,M_q_10] = bound_sub_integrals(dm,rho_psi,rho_x,rho_q,a,b);

d.M_psi_10 = M_psi_10;
d.M_x_10 = M_x_10;
d.M_q_10 = M_q_10;

abs_tol = 1e-17;
[N_x_10,err_x_10] = N_nodes(rho_x,M_x_10,abs_tol);
d.N_x_10 = N_x_10;
d.err_x_10 = err_x_10;

abs_tol = 1e-17;
[N_psi_10,err_psi_10] = N_nodes(rho_psi,M_psi_10,abs_tol);
d.N_psi_10 = N_psi_10;
d.err_psi_10 = err_psi_10;

```



```

[N_q_10,err_q_10] = N_nodes(rho_q,M_q_10,abs_tol);
d.N_q_10 = N_q_10;
d.err_q_10 = err_q_10;

%-----
% get interpolation coefficients when ntilde = 0
% for 10 integrands
%-----

a_q = a;
b_q = b;
a_psi = nm(0);
b_psi = nm(1);

ntilde = 0;
fun = (q,psi)(numer(N_x,err_x,q*(b_q-a_q)/2+(a_q+b_q)/2,...
    psi*(b_psi-a_psi)/2+(a_psi+b_psi)/2,ntilde));

% interpolation coefficients
t1 = tic;
cf10 = cf_biv_cheby(N_q_10,N_psi_10,10,fun);
d.cf10_time = toc(t1);
d.cf10 = cf10;

%-----
% Find bound on numerator when ntilde = 0
%-----

ntilde = 0;

a_psi = inf(nm('0.5'));
b_psi = 1;

time1 = tic;
[M_psi,M_x,M_q] = bound_numer...
    (dm,rho_psi,rho_x,rho_q,q_min,q_max,a_psi,b_psi,ntilde);
toc(time1)

```

```

abs_tol = 1e-17;
[N_x,err_x] = N_nodes(rho_x,M_x,abs_tol);

abs_tol = 1e-17;
[N_psi,err_psi] = N_nodes(rho_psi,M_psi,abs_tol);

[N_q,err_q] = N_nodes(rho_q,M_q,abs_tol);

d.bd_n1_time = toc(time1);

d.M_psi_n0 = M_psi;
d.M_q_n0 = M_q;
d.M_x_n0 = M_x;
d.N_psi_n0 = N_psi;
d.N_q_n0 = N_q;
d.N_x_n0 = N_x;
d.err_psi_n0 = err_psi;
d.err_q_n0 = err_q;
d.err_x_n0 = err_x;

%-----
% get interpolation coefficients when ntilde = 0
%-----

fun = (q,psi)(integrand_numer(N_x,err_x,q*(b_q-a_q)/2+(a_q+b_q)/2,...
    psi*(b_psi-a_psi)/2+(a_psi+b_psi)/2,ntilde));

% interpolation coefficients
tstart = tic;
cfn0 = cf_biv_cheby(N_q,N_psi,6,fun);
d.cf0_time = toc(t1);
d.cfn0 = cfn0;
d.cfn0_time = toc(tstart);

%-----
% save data

```

```
%
%-----

d.total_time = toc(total_time);
d.date = date;

saveit(curr_dir,'interval_arithmetic',d,file_name,'data_final');
```

3.28 kappa_lemma.m

```
% TODO
curr_dir = local_startup;
```

Now

$$\kappa^2(k) = \left(\frac{\pi^2}{K^2(k)} \right) \left(\frac{7}{20} \right) \left(\frac{A(k) - B(k)}{C(k) + D(k)} \right),$$

where

$$\begin{aligned} A(k) &:= 2(k^4 - k^2 + 1)E(k) \\ B(k) &:= (1 - k^2)(2 - k^2)K(k) \\ C(k) &:= (-2 + 3k^2 + 3k^4 - 2k^6)E(k) \\ D(k) &:= (k^6 + k^4 - 4k^2 + 2)K(k), \end{aligned}$$

where $K(k)$ and $E(k)$ are respectively the complete elliptic integrals of the first and second kind.

The derivative of $\kappa^2(k)$ is given by,

$$\begin{aligned} \frac{\partial}{\partial k} \kappa^2(k) &= \left(\frac{-2\pi K'(k)}{K^3(k)} \right) \left(\frac{7}{20} \right) \left(\frac{A(k) - B(k)}{C(k) + D(k)} \right) \\ &+ \left(\frac{\pi^2}{K^2(k)} \right) \left(\frac{7}{20} \right) \left(\frac{(C(k) + D(k))(A'(k) - B'(k)) - (A(k) - B(k))(C'(k) + D'(k))}{(C(k) + D(k))^2} \right), \end{aligned}$$

where,

$$\begin{aligned} A'(k) &= 2(4k^3 - 2k)E(k) + 2(k^4 - k^2 + 1)E'(k), \\ B'(k) &= -2k(2 - k^2)K(k) - 2k(1 - k^2)K(k) + (1 - k^2)(2 - k^2)K'(k), \\ C'(k) &= (6k + 12k^3 - 12k^5)E(k) + (-2 + 3k^2 + 3k^4 - 2k^6)E'(k), \\ D'(k) &= (6k^5 + 4k^3 - 8k)K(k) + (k^6 + k^4 - 4k^2 + 2)K'(k). \end{aligned}$$

The elliptic integrals have derivatives,

$$\begin{aligned} K'(k) &= \frac{E(k)}{k(1 - k^2)} - \frac{K(k)}{k} \\ E'(k) &= \frac{E(k) - K(k)}{k}. \end{aligned}$$

%%

pie = nm('pi');

% k interval ends

```
s0 = linspace(0.9,0.93,100);
s1 = linspace(0.93,0.99,100);
s2 = linspace(0.99,0.999,100);
s3 = linspace(0.999,0.9999,100);
s4 = linspace(0.9999,0.99999,100);
s5 = linspace(0.99999,0.999999,100);
s6 = linspace(0.999999,0.9999999,100);
k = [s0, s1, s2, s3, s4, s5, s6];
% make k intervals from end poitns
k = nm(k(1:end-1),k(2:end));
```

% complete elliptic integral of the first kind

```
K = elliptic_integral(k,1);
```

% complete elliptic integral of the second kind

```
E = elliptic_integral(k,2);
```

% derivatives of K and E with resepct to k

```
Kp = E./(k.*(1-k.^2))-K./k;
```

```
Ep = (E-K)./k;
```

% auxiliary functions

```

A = 2*(k.^4-k.^2+1).*E;

B = (1-k.^2).*(2-k.^2).*K;

C = (-2+3*k.^2+3*k.^4-2*k.^6).*E;

D = (k.^6+k.^4-4*k.^2+2).*K;

% derivatives of auxiliary functions with respect to k
Ap = 2*(4*k.^3-2*k).*E+2*(k.^4-k.^2+1).*Ep;

Bp = -2*k.*(2-k.^2).*K-2*k.*(1-k.^2).*K+(1-k.^2).*(2-k.^2).*Kp;

Cp = (6*k+12*k.^3-12*k.^5).*E+(-2+3*k.^2+3*k.^4-2*k.^6).*Ep;

Dp = (6*k.^5+4*k.^3-8*k).*K+(k.^6+k.^4-4*k.^2+2).*Kp;

% derivative of kappa^2 with respect to k
kappa2p = ((-2*pie^2*Kp)./K.^3)*(nm(7)/20).*((A-B)./(C+D)) ...
          + (pie^2./K.^2)*(nm(7)/20).*(((C+D).*(Ap-Bp)-(A-B).*(Cp+Dp))./(C+D).^2);

% check that NaN is not present
if sum(isnan(kappa2p)) > 0
    error('NaN present');
end

% find maximum value of the derivative of kappa^2 on the k intervals
mx = max(sup(kappa2p));

% check that mx < 0 implying that kappa^2 is monotone decreasing
if mx >= 0
    error('failed to verify monotonicity');
end

% find the left end of the k intervals
kleft = min(inf(k));
% find the right end of the k intervals
kright = max(sup(k));

```

```

fprintf('\nThe derivative of kappa^2 with respect to k is <= ');
disp(mx);
fprintf(' for k in the interval with left end point,');
disp(kleft);
fprintf('and right end point, ');
disp(kright);
fprintf('\n\n');

```

3.29 kappa_of_k.m

```

function kappa = kappa_of_k(k)
% function kappa = kappa_of_k(k)
%
% Returns kappa(k)

```

From

$$\left(\frac{K(k)\mathcal{G}(k)}{\pi} \right)^2 = \frac{7}{20} \frac{2(k^4 - k^2 + 1)E(k) - (1 - k^2)(2 - k^2)K(k)}{(-2 + 3k^2 + 3k^4 - 2k^6)E(k) + (k^6 + k^4 - 4k^2 + 2)K(k)}$$

we may determine $\kappa = \mathcal{G}(k)$.

```

pie = nm('pi');
% elliptic integrals
K = elliptic_integral(k,1);
E = elliptic_integral(k,2);
% KmE = elliptic_integral(k,3)

```

```

% kappa(k)
k2 = k.*k;
k4 = k2.*k2;
k6 = k2.*k4;
c1 = 2*(k4-k2+1).*E-(1-k2).*(2-k2).*K;
c2 = (-2+3*k2+3*k4-2*k6).*E+(k6+k4-4*k2+2).*K;
kappa = pie*sqrt(7*c1./(20*c2))./K;

```

3.30 lambda_xi.m

```

function [f,fd,fdd,g,gd,gdd] = lambda_xi(q,omega,omega_prime,psi,ntilde)
%
% out = lambda_xi(q,omega,ntilde,psi_tilde)
%
% Returns in the first three components
% xi(ntilde*omega_prime+1i*psi*omega_prime)
% and its first two derviations with respect to psi. Returns in the next three
% components 1i*c*lambda_0(ntilde*omega+1i*psi*omega_prime) where c is a
% real, nonzero constant.

```

Now

$$\begin{aligned}
\wp'(z) &= -\frac{\sigma(2z)}{\sigma^4(z)} \\
\sigma(z) &= \frac{2\omega}{\pi} e^{\eta_1 z^2/2\omega} \vartheta_1(\pi z/2\omega)/\vartheta_1'(0) \\
\implies \wp'(z) &= -\frac{(\pi\vartheta_1'(0))^3}{8\omega^3} \frac{\vartheta_1(\pi z/\omega)}{\vartheta_1^4(\pi z/2\omega)}
\end{aligned}$$

```

%%

```

```

% pi
pie = nm('pi');

```

```

%  $\vartheta'_1(0)$ 
v0 = theta_vec_z(q,0,1);

% number of derivatives to take
m = 2;

%  $\vartheta_1(\pi z/\omega)$ 
con = pie./(omega);
z = con.*(ntilde*omega+1i*psi.*omega_prime);
vn = theta_vec_z(q,z,m);

%  $\vartheta_1(\pi z/(2\omega))$ 
con = pie./(2*omega);
z = con.*(ntilde*omega+1i*psi.*omega_prime);
vd = theta_vec_z(q,z,m);

%  $\wp'(z)$ 
gdd = -v0(:,2).^3.*pie^3.*vn(:,1)./(8*omega.^3.*vd(:,1).^4);
%  $\frac{\partial^2}{\partial \psi^2} \omega \zeta(\tilde{n}\omega + i\psi\omega')$ 
gdd = 2*1i*omega_prime.^2.*omega.*gdd;

%  $\xi(\tilde{n} + i\psi\omega')$ 
g = xi_q_psi(q,psi,ntilde);
g = g.';

%  $\frac{\partial}{\partial \psi} \xi(\tilde{n} + i\psi\omega')$ 
gd = xi_der_q_psi(q,psi,ntilde);
gd = gd.';

% auxiliary quantities

A = ((pie./omega).*vd(:,1).^4.*vn(:,2)-(2*pie./omega).*vn(:,1).*vd(:,1).^3.*vd(:,2)));

Ad = ( (4*pie./omega).*vd(:,1).^3.*vd(:,2).*vn(:,2)*(pie./(2*omega)) ...
+ (pie./omega).*vd(:,1).^4.*vn(:,3).*(pie./omega) ...

```



```

- (2*pi./omega).*vn(:,2).*(pie./omega).*vd(:,1).^3.*vd(:,2) ...
-(2*pi./omega).*vn(:,1).*3.*vd(:,1).^2.*vd(:,2).*(pie./(2*omega)).*vd(:,2)-
...
-(2*pi./omega).*vn(:,1).*vd(:,1).^3.*vd(:,3).*(pie./(2*omega)));

B = vd(:,1).^8;

Bd = 8*vd(:,1).^7.*vd(:,2).*(pie./(2*omega));

%  $ic\lambda_0(\tilde{n}\omega + i\psi\omega')$ 
f = 1i*vn(:,1)./vd(:,1).^4;

%  $\frac{\partial}{\partial\psi}ic\lambda_0(\tilde{n}\omega + i\psi\omega')$ 
fd = A./B;
fd = 1i*fd.*(1i*omega_prime);

%  $\frac{\partial^2}{\partial\psi^2}ic\lambda_0(\tilde{n}\omega + i\psi\omega')$ 
fdd = (B.*Ad-A.*Bd)./B.^2;
fdd = 1i*fdd.*(1i*omega_prime).^2;

```

3.31 lemma_qk.m

```

curr_dir = local_startup;

pie = nm('pi');

% k interval ends
s0 = linspace(0.9,0.93,400);
s1 = linspace(0.93,0.99,400);

```

```

s2 = linspace(0.99,0.999,400);
s3 = linspace(0.999,0.9999,400);
s4 = linspace(0.9999,0.99999,400);
s5 = linspace(0.99999,0.999999,400);
s6 = linspace(0.999999,0.9999999,400);
k = [s0, s1, s2, s3, s4, s5, s6];
% make k intervals from end poitns
k = nm(k(1:end-1),k(2:end));

% complete elliptic integral of the first kind
K = elliptic_integral(k,1);
% complete elliptic integral of the second kind
E = elliptic_integral(k,2);

% complete elliptic integral of the first kind
K2 = elliptic_integral(1-k.^2,1);
% complete elliptic integral of the second kind
E2 = elliptic_integral(1-k.^2,2);

% derivatives of K and E with resepect to k
Kp = E./(k.*(1-k.^2))-K./k;
Ep = (E-K)./k;

% derivatives of K2 and E2 with resepect to k
K2p = -2*k.*(E2./((1-k.^2).*(1-(1-k.^2).^2))-K2./(1-k.^2));
E2p = -2*k.*(E2-K2)./(1-k.^2);

% part of derivative of q(k) with respect to k
T = -2*k.*K.*K2p-K2.*Kp;

mx = max(sup(T));
% find the left end of the k intervals
kleft = min(inf(k));
% find the right end of the k intervals
kright = max(sup(k));

% mx negative indicates that the derivative of q(k) with respect
% to k is strictly increasing on interval [kleft,kright]

```

3.32 local_startup.m

```
function curr_dir = local_startup

clear all; close all; beep off; clc; curr_dir = cd;
%% startup commands
cd('..');
cd('..');
startup('intlabb',',','start matlabpool','off');
format long;
clc;
cd(curr_dir);

% display type
intvalinit('DisplayMidRad');
% intvalinit('DisplayInfSup');
curr_dir = cd;
```

3.33 local_startup_batch.m

```
function local_startup_batch(curr_dir)

%% startup commands
cd('..');
cd('..');
startup('intlabb',',','start matlabpool','off');
format long;

cd(curr_dir);

% display type
% intvalinit('DisplayMidRad');
intvalinit('DisplayInfSup');
clc;
```

3.34 lower_bound.m

```
function [dm,rho_x] = lower_bound(frac,q_min,q_max,num_steps,steps)
% frac must be strictly between 0 and 1 (for choosing rho_x)

% Get a lower bound on theta function that shows up in the denominator
% in the definition of v(x)

temp = lower_bound_local(q_min,q_max,num_steps,frac,steps);

dm = min(temp);

pie = nm('pi');
rho_con = -frac*log(q_max)/pie;
rho_x = rho_con+sqrt(rho_con^2+1);

%-----
% lower_bound_local
%-----
function out = lower_bound_local(q_min,q_max,num_steps,frac,steps)

pie = nm('pi');
% parity and mirror symmetry of  $\vartheta_1$  only requires we go
% to pi/2 instead of 2 pi.
half = nm('0.5');
con = (half*pie/steps);

ind = 0:1:steps;
theta = nm(ind(1:end-1)*con,ind(2:end)*con);

q_del = (q_max-q_min)/num_steps;
psi = 0; ntilde = 0;
fun = (q,x)(theta_vec(q,psi,x,0,ntilde));
```

```

out = 1000;
for j = 1:num_steps-1

%   if mod(j,10) == 0
%       fprintf('percent done = %% %4.4g\n',100*j/num_steps);
%   end

    q = q_min + nm((j-1)*q_del,j*q_del);
    rho_x = get_rho(q,frac);

    x = half*(rho_x*exp(1i*theta)+exp(-1i*theta)/rho_x);
    out = min(out, min(abs(fun(q,x))));
    if out == 0
        error('problem')
    end

end

if isnan(out)
    error('NaN given for error bound');
end

%-----
% get rho
%-----
function rho_x = get_rho(q,frac)

%
%% compute rho for ellipse used in getting Chebyshev bounds for x variable
%
```

To interpolate in $x \in [-1, 1]$ we need a bound on the integrands. The integrands are analytic in and on an ellipse that does not intersect zeros of $\vartheta_1(\pi(x \pm i\omega')/2)$. Now the zeros of ϑ_1 are the set $\{m\pi + n\pi i\omega'/\omega | m, n \in \mathbb{N}\}$. Then from

$$\frac{\pi}{2\omega}(\omega x \pm i\omega') = m\pi + n\pi i\omega'/\omega \quad (3.7)$$

we find

$$\Im(x) < \omega'/\omega. \quad (3.8)$$

is required.

Now if $0 < c < \omega'/\omega$ is the height of the top of the ellipse E_ρ , then

$$\frac{1}{2}(\rho - 1/\rho) = c, \quad (3.9)$$

then

$$\rho = c + \sqrt{c^2 + 1}. \quad (3.10)$$

```
pie = nm('pi');
rho_con = -frac*log(q)/pie;
rho_x = rho_con+sqrt(rho_con^2+1);
```

3.35 lower_bound_unstable.m

```
function [dm,rho_x,q_min_out,q_max_out] = lower_bound_unstable
```

```
temp = zeros(5,1);
frac = nm('0.9'); % strictly between 0 and 1 (for choosing rho_x)
```

```
q_min = nm('0.1');
q_max = nm('0.15');
q_max_out = q_max;
num_steps = 100;
temp(1) = lower_bound_local(q_min,q_max,num_steps,frac);
```

```

q_min = nm('0.01');
q_max = nm('0.1');
num_steps = 100;
temp(2) = lower_bound_local(q_min,q_max,num_steps,frac);

q_min = nm('0.001');
q_max = nm('0.01');
num_steps = 100;
temp(3) = lower_bound_local(q_min,q_max,num_steps,frac);

q_min = nm('0.0001');
q_max = nm('0.001');
num_steps = 100;
temp(4) = lower_bound_local(q_min,q_max,num_steps,frac);

q_min = nm('1e-7');
q_max = nm('0.0001');
q_min_out = q_min;
num_steps = 100;
temp(5) = lower_bound_local(q_min,q_max,num_steps,frac);

dm = min(temp);

pie = nm('pi');
rho_con = -frac*log(q_max)/pie;
rho_x = rho_con+sqrt(rho_con^2+1);

%-----
% lower_bound_local
%-----
function out = lower_bound_local(q_min,q_max,num_steps,frac)

steps = 8000;
pie = nm('pi');
% parity and mirror symmetry of  $\vartheta_1$  only requires we go
% to pi/2 instead of 2 pi.

```

```

half = nm('0.5');
con = (half*pie/steps);

ind = 0:1:steps;
theta = nm(ind(1:end-1)*con,ind(2:end)*con);

q_del = (q_max-q_min)/num_steps;
psi = 0; ntilde = 0;
fun = (q,x)(theta_vec(q,psi,x,0,ntilde));

out = 1000;
for j = 1:num_steps-1

    q = q_min + nm((j-1)*q_del,j*q_del);
    rho_x = get_rho(q,frac);

    x = half*(rho_x*exp(1i*theta)+exp(-1i*theta)/rho_x);
    out = min(out, min(abs(fun(q,x))));

end

% check if lower bound is not helpful
if out == 0
    error('lower bound is 0');
end

% check that out is not NaN
if (out < 10)
    error('error apparent')
end

%-----
% get rho
%-----
function rho_x = get_rho(q,frac)

```



```
%
%% compute rho for ellipse used in getting Chebyshev bounds for x variable
%
```

To interpolate in $x \in [-1, 1]$ we need a bound on the integrands. The integrands are analytic in and on an ellipse that does not intersect zeros of $\vartheta_1(\pi(x \pm i\omega')/2)$. Now the zeros of ϑ_1 are the set $\{m\pi + n\pi i\omega'/\omega | m, n \in \mathbb{N}\}$. Then from

$$\frac{\pi}{2\omega}(\omega x \pm i\omega') = m\pi + n\pi i\omega'/\omega \quad (3.11)$$

we find

$$\Im(x) < \omega'/\omega. \quad (3.12)$$

is required.

Now if $0 < c < \omega'/\omega$ is the height of the top of the ellipse E_ρ , then

$$\frac{1}{2}(\rho - 1/\rho) = c, \quad (3.13)$$

then

$$\rho = c + \sqrt{c^2 + 1}. \quad (3.14)$$

```
pie = nm('pi');
rho_con = -frac*log(q)/pie;
rho_x = rho_con+sqrt(rho_con^2+1);
```

3.36 lower_instability_interpolation.m

```

% verify instability lower
clear all; curr_dir = local_startup;

pie = nm('pi');

tstart = tic;
[dm,rho_x,q_min_out,q_max_out] = lower_bound_unstable;
d.dm_time = toc(tstart);
d.dm = dm;
d.rho_x = rho_x;

%
% choices of q
%

num = 7;
KL = zeros(num,1);
RK = zeros(num,1);
MN = zeros(num,1);
time = zeros(num,1);

cnt = 0;

cnt = cnt + 1;
stent1 = nm('0.122'); % qmin
stent2 = nm('0.139'); % qmax
stent3 = [linspace(inf(nm('0.93')),sup(nm('0.9422')),100),sup(nm('0.9423'))]; %-
k points
stent4 = 1e-4;

cnt = cnt + 1;
stent1 = nm('0.1');
stent2 = nm('0.2');
stent3 = [ 0.899, 0.93];
stent4 = 1e-4;

cnt = cnt + 1;
stent1 = nm('0.05');

```

```

stcnt2 = nm('0.11');
stcnt3 = [ 0.75, 0.8 0.9];
stcnt4 = 1e-4;

cnt = cnt + 1;
stcnt1 = nm('0.01');
stcnt2 = nm('0.06');
stcnt3 = linspace(0.4,0.751,10);
stcnt4 = 1e-4;

cnt = cnt + 1;
stcnt1 = nm('0.005');
stcnt2 = nm('0.011');
stcnt3 = [0.3 0.35 0.4 ];
stcnt4 = 1e-4;

cnt = cnt + 1;
stcnt1 = nm('0.003');
stcnt2 = nm('0.006');
stcnt3 = [0.24 0.25 0.26 0.27 0.28 0.29 0.3 ];
stcnt4 = 1e-4;

cnt = cnt + 1;
stcnt1 = nm('0.002');
stcnt2 = nm('0.0038');
stcnt3 = [ 0.2 0.21 0.22 0.23 0.24 ];
stcnt4 = 1e-4;

total_time_start = tic;
for ind = 1:num

    fprintf('Lower instability interpolation - percent done: %3.3g\n',100*(ind-
1)/7);

    tstart = tic;
    % interpolate
    d.q_min = stind1;
    d.q_max = stind2;
    d = instability_interpolation(d,stind1,stind2);

    file_name = ['d_lower_unstable_',num2str(ind)];

```

```

saveit(curr_dir,'interval_arithmetic',d,file_name,'data_final');

% divide up into k intervals
kpts = stind3;
left_k = zeros(length(kpts)-1,1);
right_k = zeros(length(kpts)-1,1);
min_quot = zeros(length(kpts)-1,1);

% verify stability
eps = stind4;
for j = 1:length(kpts)-1
    k = linspace(kpts(j)-eps,kpts(j+1)+eps,1000);
    [left_k(j),right_k(j),min_quot(j)] = verify_instability_lower(d,k);
end

% check for gap in inner k intervals
for j = 1:length(left_k)-1
    if right_k(j) < left_k(j+1)
        error('gap present');
    end
end

% record data
KL(ind) = left_k(1);
RK(ind) = right_k(end);
MN(ind) = min(min_quot);

% error check for gap between outer k intervals
for j = 1:ind-1
    if RK(j+1) < KL(j)
        error('gap present');
    end
end

% record time
time(ind) = toc(tstart);

end

data.total_time = toc(total_time_start);

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Left_K = KL(end);
Right_K = RK(1);
MNtotal = min(MN);
```

```
file_name = 'instability_result_lower';
data.Left_K = Left_K;
data.Right_K = Right_K;
data.KL = KL;
data.RK = RK;
data.MNtotal = MNtotal;
data.MN = MN;
data.st = st;
data.time = time;
saveit(curr_dir,'interval_arithmetic',data,file_name,'data_final');
```

3.37 numer.m

```
function out = numer(Nx,err,q,psi,ntilde)
% out = numer(Nx,err,q,psi,ntilde)
%
%
```

Definition of λ_1

$$\lambda_1 = \frac{\int_{-1}^1 \left[\frac{\partial}{\partial x} v(\omega x) + \frac{1}{\omega^2} \frac{\partial^3}{\partial x^3} v(\omega x) \right] \frac{\partial^2}{\partial x^2} \bar{v}(\omega x) dx}{\omega^2 \int_{-1}^1 v(\omega x) \frac{\partial}{\partial x} \bar{v}(\omega x) dx}. \quad (3.15)$$

Now

$$\begin{aligned} v(x) &= w(x)e^{\gamma x} \\ v'(x) &= w'(x)e^{\gamma x} + \gamma w(x)e^{\gamma x} \\ v''(x) &= w''(x)e^{\gamma x} + 2\gamma w'(x)e^{\gamma x} + \gamma^2 w(x)e^{\gamma x} \\ v'''(x) &= w'''(x)e^{\gamma x} + 3\gamma w''(x)e^{\gamma x} + 3\gamma^2 w'(x)e^{\gamma x} + \gamma^3 w(x)e^{\gamma x}. \end{aligned}$$

Then

$$\begin{aligned} v(\omega x)\bar{v}'(\omega x) &= w(\omega x)\bar{w}'(\omega x) + (i\xi)w(\omega x)\bar{w}(\omega x), \\ v(\omega x)\bar{v}''(\omega x) &= \sum_{n=0}^3 \gamma^n c_n(x), \\ v'''(\omega x)\bar{v}''(\omega x) &= \sum_{n=0}^5 \gamma^n h_n(x), \end{aligned} \quad (3.16)$$

where

$$\begin{aligned} c_0(x) &:= w'(\omega x)\bar{w}''(\omega x) \\ c_1(x) &:= w(\omega x)\bar{w}''(\omega x) - 2w'(\omega x)\bar{w}'(\omega x) \\ c_2(x) &:= w'(\omega x)\bar{w}(\omega x) - 2w(\omega x)\bar{w}'(\omega x) \\ c_3(x) &:= w(\omega x)\bar{w}(\omega x) \\ h_0(x) &:= w'''(\omega x)\bar{w}''(\omega x) \\ h_1(x) &:= 3w''(\omega x)\bar{w}''(\omega x) - 2w'''(\omega x)\bar{w}'(\omega x) \\ h_2(x) &:= w'''(\omega x)\bar{w}(\omega x) - 6w''(\omega x)\bar{w}'(\omega x) + 3w'(\omega x)\bar{w}''(\omega x) \\ h_3(x) &:= 3w''(\omega x)\bar{w}(\omega x) - 6w'(\omega x)\bar{w}'(\omega x) + w(\omega x)\bar{w}''(\omega x) \\ h_4(x) &:= 3w'(\omega x)\bar{w}(\omega x) - 2w(\omega x)\bar{w}'(\omega x) \\ h_5(x) &:= w(\omega x)\bar{w}(\omega x). \end{aligned} \quad (3.17)$$

%%

% _____
% Get theta and derivatives

```

% -----

% constants
pie = nm('pi');
half = nm(1)/2;

% interpolation points in x for integration
theta = ((0:1:Nx)+half)*pie/(Nx+1);
x = cos(theta);

% denominator term (alpha = 0)
J = theta_vec(q,0,x,3,0);
J = repmat(J,[1 length(psi)]);

E0 = 1./J(:,1);
E1 = -J(:,2)./J(:,1).^2;
E2 = 2*J(:,2).^2./J(:,1).^3-J(:,3)./J(:,1).^2;
E3 = -6*J(:,2).^3./J(:,1).^4+6*J(:,2).*J(:,3)./J(:,1).^3-J(:,4)./J(:,1).^2;

B0 = E0.*E0;
B1 = 2*E0.*E1;
B2 = 2*(E0.*E2+E1.*E1);
B3 = 2*(E3.*E0+3*E1.*E2);

% numerator term (alpha = ntilde*omega + 1i*psi*omega')
L = theta_vec(q,psi,x,4,ntilde);

A0 = L(:,1).*L(:,1);
A1 = 2*L(:,1).*L(:,2);
A2 = 2*(L(:,1).*L(:,3)+L(:,2).*L(:,2));
A3 = 2*(L(:,4).*L(:,1)+3*L(:,2).*L(:,3));

w0 = A0.*B0;
w1 = A0.*B1+A1.*B0;
w2 = A0.*B2+2*A1.*B1+A2.*B0;
w3 = A0.*B3+3*A1.*B2+3*A2.*B1+ A3.*B0;

% -----

```

```

% functions in expansions
% -----

%  $v(\omega x)\bar{v}''(\omega x) = \sum_{n=0}^3 \gamma^n c_n(x)$ 
c0 = w1.*conj(w2);
c1 = w0.*conj(w2)-2*w1.*conj(w1);
c2 = w1.*conj(w0)-2*w0.*conj(w1);
c3 = w0.*conj(w0);

%  $v'''(\omega x)\bar{v}''(\omega x) = \sum_{n=0}^5 \gamma^n h_n(x)$ 
h0 = w3.*conj(w2);
h1 = 3*w2.*conj(w2)-2*w3.*conj(w1);
h2 = w3.*conj(w0)-6*w2.*conj(w1)+3*w1.*conj(w2);
h3 = 3*w2.*conj(w0)-6*w1.*conj(w1)+w0.*conj(w2);
h4 = 3*w1.*conj(w0)-2*w0.*conj(w1);
h5 = w0.*conj(w0);

% Chebyshev polynomials evaluated at the interpolation points
Tx = cos(theta.*(0:2:Nx));

% Chebyshev coefficients
cf_c0 = 2*c0.*Tx/(Nx+1);
cf_c0(:,1) = cf_c0(:,1)/2;

cf_c1 = 2*c1.*Tx/(Nx+1);
cf_c1(:,1) = cf_c1(:,1)/2;

cf_c2 = 2*c2.*Tx/(Nx+1);
cf_c2(:,1) = cf_c2(:,1)/2;

cf_c3 = 2*c3.*Tx/(Nx+1);
cf_c3(:,1) = cf_c3(:,1)/2;

cf_h0 = 2*h0.*Tx/(Nx+1);
cf_h0(:,1) = cf_h0(:,1)/2;

cf_h1 = 2*h1.*Tx/(Nx+1);
cf_h1(:,1) = cf_h1(:,1)/2;

```



```
cf_h2 = 2*h2.*Tx/(Nx+1);
cf_h2(:,1) = cf_h2(:,1)/2;
```

```
cf_h3 = 2*h3.*Tx/(Nx+1);
cf_h3(:,1) = cf_h3(:,1)/2;
```

```
cf_h4 = 2*h4.*Tx/(Nx+1);
cf_h4(:,1) = cf_h4(:,1)/2;
```

```
cf_h5 = 2*h5.*Tx/(Nx+1);
cf_h5(:,1) = cf_h5(:,1)/2;
```

```
out = nm(zeros(length(psi),1,10));
out(:,1) = 2*cf_c0*(1./(1-(0:2:Nx).^2)).';
out(:,2) = 2*cf_c1*(1./(1-(0:2:Nx).^2)).';
out(:,3) = 2*cf_c2*(1./(1-(0:2:Nx).^2)).';
out(:,4) = 2*cf_c3*(1./(1-(0:2:Nx).^2)).';
out(:,5) = 2*cf_h0*(1./(1-(0:2:Nx).^2)).';
out(:,6) = 2*cf_h1*(1./(1-(0:2:Nx).^2)).';
out(:,7) = 2*cf_h2*(1./(1-(0:2:Nx).^2)).';
out(:,8) = 2*cf_h3*(1./(1-(0:2:Nx).^2)).';
out(:,9) = 2*cf_h4*(1./(1-(0:2:Nx).^2)).';
out(:,10) = 2*cf_h5*(1./(1-(0:2:Nx).^2)).';
```

```
% add integration ( in x ) error
out = out + 2*(nm(-err,err)+1i*nm(-err,err));
```

3.38 save_figure.m

```
% save figure in BFZstudy folder
```

```

sf.name = 'intervalarithmetic';
try
    savefig(sf,cd,'interval_arithmetic',gcf,p)
catch
    savefig(sf,cd,'interval_arithmetic',gcf)
end

```

3.39 simplicity.m

```

clear all; close all; beep off; clc; curr_dir = cd;
%% startup commands
cd('..');
cd('..');
startup('intl'ab','start matlabpool','off');
format long;
clc;
cd(curr_dir);

% display type
intvalinit('DisplayMidRad');
% intvalinit('DisplayInfSup');

%
%% parameters
%

st1 = linspace(inf(nm('0.942')),sup(nm('0.97')),28001);
st2 = linspace(inf(nm('0.97')),sup(nm('0.99')),20001);

st3 = linspace(inf(nm('0.99')),sup(nm('0.995')),10000);
st4 = linspace(inf(nm('0.995')),sup(nm('0.999')),10000);
st5 = linspace(inf(nm('0.999')),sup(nm('0.9995')),10000);
st6 = linspace(inf(nm('0.9995')),sup(nm('0.9999')),10000);
st7 = linspace(inf(nm('0.9999')),sup(nm('0.99992')),10000);
st8 = linspace(inf(nm('0.99992')),sup(nm('0.99993')),10000);
st9 = linspace(inf(nm('0.99993')),sup(nm('0.999934')),10000);
st10 = linspace(inf(nm('0.999934')),sup(nm('0.999938')),10000);

```

```

st11 = linspace(inf(nm('0.999938')),sup(nm('0.99999')),20000);
st12 = linspace(inf(nm('0.99999')),sup(nm('0.999998')),20000);
st13 = linspace(inf(nm('0.999998')),sup(nm('0.999998134')),2500);
st14 = linspace(inf(nm('0.999998134')),sup(nm('0.999998267')),2500);
st15 = linspace(inf(nm('0.999998267')),sup(nm('0.9999984')),2500);

out = zeros(length(st),1);
parfor j = 1:length(out)
    out(j) = simplicity_aux(stj,curr_dir);
end

% 1 indicates success and 0 indicates failure in verifying simplicity
for j = 1:length(out)
    if out(j) == 0
        error('Failed to verify simplicity')
    end
end
end

```

3.40 simplicity_aux.m

```
function out = simplicity_aux(k_vals,curr_dir)

out = 0;

%% startup commands

local_startup_batch(curr_dir);

for j = 1:length(k_vals)-1

    k = nm(k_vals(j),k_vals(j+1));

    %
    %% derived constants
    %

    p.k = k;
    pie = nm('pi');
    kappa = kappa_of_k(k);
    elipk = elliptic_integral(k,1);
    elipk2 = elliptic_integral(sqrt(1-k.^2),1);
    omega = pie./kappa;
    omega_prime = elipk2*pie./(elipk.*kappa);
    q = exp(-pie*elipk2./elipk);

    %
    % -----
    %

    ntilde = 1;

    num = 100;
    L = 20;  %  $\beta_0 = 0.95$ 
    L2 = 1;  %  $\beta \in [0, \omega']$ 
```

```

psi_x = nm(0:1:num)/(L2*num)+nm(L2-1)/L2;
psi_x = nm(psi_x(1:end-1),psi_x(2:end)); % make intervals

psi_y = nm(0:1:num)/(L*num)+nm(L-1)/L;
psi_y = nm(psi_y(1:end-1),psi_y(2:end)); % make intervals

num = num -1;

% get components of h(x,y)
[f1,fd1,fdd1,g1,gd1,gdd1] = lambda_xi(q,omega,omega_prime,psi_x,ntilde);

F1 = repmat(f1,1,num+1);
Fd1 = repmat(fd1,1,num+1);
Fdd1 = repmat(fdd1,1,num+1);
G1 = repmat(g1,1,num+1);
Gd1 = repmat(gd1,1,num+1);
Gdd1 = repmat(gdd1,1,num+1);

%
% -----
%

ntilde = 0;

% get components of h(x,y)
[f2,fd2,fdd2,g2,gd2,gdd2] = lambda_xi(q,omega,omega_prime,psi_y,ntilde);
f2 = f2.'; fd2 = fd2.'; fdd2 = fdd2.'; g2 = g2.'; gd2 = gd2.'; gdd2 = gdd2.';

F2 = repmat(f2,num+1,1);
Fd2 = repmat(fd2,num+1,1);
Fdd2 = repmat(fdd2,num+1,1);
G2 = repmat(g2,num+1,1);
Gd2 = repmat(gd2,num+1,1);
Gdd2 = repmat(gdd2,num+1,1);

%
% -----

```

```

%

% Show that  $\frac{\partial}{\partial \psi} \xi(\omega + i\psi\omega') > 0$  for  $\psi \in [0, 1]$ .

con = 1;

T = con*(F1+F2);
Tx = con*(Fd1);
Txx = con*(Fdd1);
% Txy = 0
Ty = con*(Fd2);
Tyy = con*(Fdd2);

S = G1+G2-2*pie;
Sx = Gd1;
% Sxy = 0
Sxx = Gdd1;
Sy = Gd2;
Syy = Gdd2;

% h = T.^2+S.^2;
% hx = 2*T.*Tx+2*S.*Sx;
hxx = 2*Tx.^2+2*T.*Txx+2*Sx.^2+2*S.*Sxx;
% hy = 2*T.*Ty+2*S.*Sy;
hyy = 2*Ty.^2+2*T.*Tyy+2*Sy.^2+2*S.*Syy;
hxy = 2*Ty.*Tx+2*Sy.*Sx;

%Del

Del = (hxx.*hyy-hxy.^2);
Del = inf(real(Del));

if abs(sum(sum(isfinite(Del)-1))) > 0
    error('Del is not finite');
end

min_Del = min(min(Del));

```

```

% hxx
hxx = inf(real(hxx));

if abs(sum(sum(isfinite(hxx)-1))) > 0
    error('hxx is not finite');
end

min_hxx = min(min(hxx));

temp1 = -4*gdd1/(2*1i*omega_prime^2*omega);
max_gdd1 = max(sup(abs(imag(temp1))));

temp2 = -4*gdd2(1)/(2*1i*omega_prime^2*omega);
gdd1_one = inf(imag(temp2));

% throw errors if we do not verify the necessary properties

if min_Del <=0
    return
end

if min_hxx <= 0
    return
end

if inf(nm(gdd1_one)-nm(max_gdd1)) <= 0
    return
end

if -inf(g2(1))-3*pie/2 <= 0
    return
end

end

```

```
out = 1;
```

3.41 strict_taylor.m

```
function out = strict_taylor(cf,q_tilde,psi_tilde,id)

% points
q_tilde = nm(q_tilde);
psi_tilde = nm(psi_tilde);
tx = acos(q_tilde).';
ty = acos(psi_tilde);

%
Nx = size(cf,1)-1; % degree of polynomial in x
Ny = size(cf,2)-1; % degree of polynomial in y
indx = 0:1:Nx;
indy = (0:1:Ny).';

zx = nm(tx)*indx; % function argument for x
zy = indy*nm(ty); % function argument for y

if id == 1 % f
    cosx = cos(zx);
    cosy = cos(zy);
    out = cosx*cf*cosy;

elseif id == 2 % f_psi
    cosx = cos(zx);
    nsiny = repmat(indy,1,length(ty)).*sin(zy);
    out = cosx*cf*(nsiny.*repmat(1./sin(ty),length(indy),1));

elseif id == 3 % f_q
```



```

    nsinx = repmat(indx,length(tx),1).*sin(zx);
    cosy = cos(zy);
    out = (repmat(1./sin(tx),1,length(indx)).*nsinx)*cf*cosy;

elseif id == 4 % f_q-q

    n2cosx = repmat(indx.^2,length(tx),1).*cos(zx);
    cosy = cos(zy);
    out = (repmat(-1./sin(tx).^2,1,length(indx)).*n2cosx)*cf*cosy;

end

```

3.42 strict_transition.m

```
function [mx_fk,mn_fpsipsi] = strict_transition(k,psi0,min_diff_q,min_diff_psi,d)
```

```

%-----
% background
%-----

```

The Taylor expansion is give by,

$$\begin{aligned}
 \backslash \text{eq} \\
 f(k, \backslash \text{psi}) \ \&= f(k_{-}, \backslash \text{psi}_{-}) + f_{\backslash k}(k_{-}, \backslash \text{psi}_{-})(k - k_{-}) + f_{\backslash \text{psi}}(k_{-}, \backslash \text{psi}_{-})(\backslash \text{psi} - \backslash \text{psi}_{-}) - \\
 + \backslash \backslash \\
 \&\frac{1}{2} \left(f_{\backslash k \backslash k}(\backslash \text{hat } k, \backslash \text{hat } \backslash \text{psi})(k - k_{-})^2 + 2 f_{\backslash \text{psi } k}(\backslash \text{hat } k, \backslash \text{hat } \backslash \text{psi})(k - k_{-})(\backslash \text{psi} - \backslash \text{psi}_{-}) + f_{\backslash \text{psi } \backslash \text{psi}}(\backslash \text{hat } k, \backslash \text{hat } \backslash \text{psi})(\backslash \text{psi} - \backslash \text{psi}_{-})^2 \right)
 \end{aligned}$$

```

%-----
% constants
%-----

k0 = k;
pie = nm('pi');

psi_tilde0 = 4*psi0-3;

% constants for transformation in q
c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;

% elliptic integrals
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
% q
q0 = exp(-pie*elipk2./elipk);
% omega0 = pie/kappa;

if inf(q0) < d.a_q
%   q0
    error('q out of range')
end
if sup(q0) > d.b_q
    error('q out of range')
end

% get theta values for q
q_tilde0 = c1_q*(q0-c2_q);

% interpolation coefficients
cf = d.cfn0;

% interpolation error
lam_n0_q = (2/pie)*log(d.N_q_n0)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n0));

```

```

err = d.err_q_n0+lam_n0_q*d.err_psi_n0;
err = nm(-err,err)+1i*nm(-err,err);

%-----
% interpolation error of interpolant derivative
%-----

% To bound the interpoaltion error coming from approximating the derivative
% of f
% with the derivative of the interpolating polynomial p, we need a bound on
% the derivative of f on a stadium. We use Cauchy's integral formula to get that
% bound.
% We already have a bound on f on a stadium of radius rho_q. We choose-
rho_q_sm and
% rho_psi_sm so that the smaller ellipse has a minor axis of half the length of
the larger one.
% Then we use Cauchy's integral formula with f on the larger stadium to get a
bound on the
% derivative of f on the smaller stadium. We then use the radius of the smaller
stadium to get
% the needed interpolation error bounds for the derivative of f.
rho_q_sm = ((d.rho_q - 1/d.rho_q)+sqrt((d.rho_q-1/d.rho_q)^2+16))/4;
rho_psi_sm = ((d.rho_psi - 1/d.rho_psi)+sqrt((d.rho_psi-1/d.rho_psi)^2+16))/4;
rho_q = d.rho_q;
rho_psi = d.rho_psi;

% Next we find the minimum distance between the larger and smaller stadiums
to
% provide a bound on the integrand in Taylor's integral formula.

% This is a bound on the derivative of f on the smaller stadium
% derived using Cauchy's integral formual
M_Dq = (1/nm(2))*sqrt(rho_q^2+1/rho_q^2)*d.M_q_n0/min_diff_q^2;
M_Dpsi = (1/nm(2))*sqrt(rho_psi^2+1/rho_psi^2)*d.M_psi_n0/min_diff_psi^2;

% This is a bound on the length of the smaller stadium
Lq = pie*sqrt(rho_q_sm^2+1/rho_q_sm^2);
% This is a lower bound on the distance between the
% stadium and the line on which we interpolate
Ddq = (rho_q_sm+1/rho_q_sm)/2-1;
eta_q = log(rho_q_sm);

```

```

err_Dq = M_Dq*Lq/(pie*Ddq*sinh(eta_q*(d.N_q_n0+1)));

% This is a bound on the length of the smaller stadium
Lpsi = pie*sqrt(rho_psi_sm^2+1/rho_psi_sm^2);
% This is a lower bound on the distance between the
% stadium and the line on which we interpolate
Ddpsi = (rho_psi_sm+1/rho_psi_sm)/2-1;
eta_psi = log(rho_psi_sm);
err_Dpsi = M_Dpsi*Lpsi/(pie*Ddpsi*sinh(eta_psi*(d.N_psi_n0+1)));

% we take the error to be the larger so we can use either variable
err_interp_der = nm(err_Dq,err_Dpsi);

% Next we find the 2-d interpolation error for the derivative of f with
% respect to q

Drhoq = (d.rho_q+1/d.rho_q)/2-1;
fc = (d.N_q_n0+1)*(d.N_q_n0+3)/Drhoq+1/Drhoq^2;
err_Dqf = fc*((pie*sqrt(rho_q^2+1/rho_q^2))*d.M_q_n0/(2*pie*sinh(log(d.rho_q)*(d.N_q_n0+1))));

% interpolation error
lam_n0_psi = (2/pie)*log(d.N_psi_n0)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_psi_n0));
err_k_q = err_interp_der+lam_n0_psi*err_Dqf;
err_k_q = nm(-err_k_q,err_k_q)+1i*nm(-err_k_q,err_k_q);

%-----
% f0_k
%-----

k = k0;

kappa = kappa_of_k(k);
K = elliptic_integral(k,1);
E = elliptic_integral(k,2);

R = elliptic_integral(sqrt(1-k^2),1);
T = elliptic_integral(sqrt(1-k^2),2);

```

```
% Derivative of complete elliptic integral of the first kind
%  $\frac{\partial}{\partial k} K(k)$  K_k = -K/k+E/(k*(1-k^2));
```

```
sqk = sqrt(1-k^2);R_k = -k*(-R/sqk+T/(k^2*sqk))/sqk;
```

```
q_k = (-pie*R_k/K + pie*K_k*R/K^2)*exp(-pie*R/K);
```

```
qtilde_q = 2/(d.b_q-d.a_q);
```

```
omega = pie/kappa;
```

```
kap_k = pie*sqrt((E*(2*k^4 - 2*k^2 + 2) + K*(-k^2 + 2)*(k^2 - 1))/(E*(-
2*k^6 + 3*k^4 + 3*k^2 - 2) + K*(k^6 + k^4 - 4*k^2 + 2)))*(E*(-2*k^6 +
3*k^4 + 3*k^2 - 2) + K*(k^6 + k^4 - 4*k^2 + 2))*((E*(2*k^4 - 2*k^2 +
2) + K*(-k^2 + 2)*(k^2 - 1))*(-E*(-12*k^5 + 12*k^3 + 6*k) - K*(6*k^5 +
4*k^3 - 8*k) - (E/(k*(-k^2 + 1)) - K/k)*(k^6 + k^4 - 4*k^2 + 2) - (E -
K)*(-2*k^6 + 3*k^4 + 3*k^2 - 2)/k)/(2*(E*(-2*k^6 + 3*k^4 + 3*k^2 - 2)
+ K*(k^6 + k^4 - 4*k^2 + 2))^2) + (E*(8*k^3 - 4*k) + 2*K*k*(-k^2 + 2)
- 2*K*k*(k^2 - 1) + (-k^2 + 2)*(k^2 - 1)*(E/(k*(-k^2 + 1)) - K/k) + (E -
K)*(2*k^4 - 2*k^2 + 2)/k)/(2*(E*(-2*k^6 + 3*k^4 + 3*k^2 - 2) + K*(k^6
+ k^4 - 4*k^2 + 2))))/(K*(E*(2*k^4 - 2*k^2 + 2) + K*(-k^2 + 2)*(k^2 -
1))) - pie*sqrt((E*(2*k^4 - 2*k^2 + 2) + K*(-k^2 + 2)*(k^2 - 1))/(E*(-2*k^6
+ 3*k^4 + 3*k^2 - 2) + K*(k^6 + k^4 - 4*k^2 + 2)))*(E/(k*(-k^2 + 1)) -
K/k)/K^2 ;kappa_k = sqrt(nm(7)/20)*kap_k;omega_k = -pi*kappa_k/kappa^2;
```

```
f1_tilde_q = strict_taylor(cf(:,1),q_tilde0,psi_tilde0,3)+err_k_q;f2_tilde_q = strict_taylor(cf(:,3),q_tilde0,psi_tild
= strict_taylor(cf(:,3),q_tilde0,psi_tilde0,1)+err;
```

```
f1_k = q_k*qtilde_q*f1_tilde_q;f2_k = q_k*qtilde_q*f2_tilde_q;
```

```
f_k = f1_k+f2_k/omega^2-2*f2*omega_k/omega^3;
```

$$f0_{\psi\psi}$$

Next we find the 2-d interpolation error for the derivative of f with respect to q

```
Drhopsi = (d.rho_psi+1/d.rho_psi)/2-1;fc = (d.N_psi_n0+1)*(d.N_psi_n0+3)/Drhopsi+1/Drhopsi^2;err_Dpsif-
= fc*((pie*sqrt(rho_psi^2+1/rho_psi^2))*d.M_psi_n0/(2*pie*sinh(log(d.rho_psi)*(d.N_psi_n0+1))));
```

```
interpolation error lam_n0_q = (2/pie)*log(d.N_q_n0)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n0));err_k
```

```

= err_interp_der+lam_n0_q*err_Dpsif;err_k_psi = nm(-err_k_psi,err_k_psi)+1i*nm(-
err_k_psi,err_k_psi);

psitilde_psi = nm(4);

f1_psi_psitilde = strict_taylor(cf(:, :, 2), q_tilde0, psi_tilde0, 2)+err_k_psi; f2_psi_psitilde =
strict_taylor(cf(:, :, 4), q_tilde0, psi_tilde0, 2)+err_k_psi;

f1_psi_psi = psitilde_psi*f1_psi_psitilde; f2_psi_psi = psitilde_psi*f2_psi_psitilde;

f_psi_psi = f1_psi_psi + f2_psi_psi/omega^2;

mx_fk = max(sup(imag(f_k))); mn_fpsipsi = min(inf(imag(f_psi_psi)));

```

3.43 strict_transition_lower.m

```

function strict_transition_lower(d,k,psi0,psiL,min_diff_q,rho_q_sm)

```

```

%-----
% constants
%-----

```

```

pie = nm('pi');

```

```

psi_tilde0 = 2*psi0-1;

```

```

% constants for transformation in q
c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
% elliptic integrals
kappa = kappa_of_k(k);
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
q0 = exp(-pie*elipk2./elipk);
omega0 = pie/kappa;

```

```

if inf(q0) < d.a_q
    error('q out of range')

```

```

end
if sup(q0) > d.b_q
    error('q out of range')
end

% get theta values for q
q_tilde0 = c1_q*(q0-c2_q);

% interpolation coefficients
cf = d.cfn1;

% interpolation error
lam_n1_q = (2/pie)*log(d.N_q_n1)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n1));
err = d.err_q_n1+lam_n1_q*d.err_psi_n1;
err = nm(-err,err)+1i*nm(-err,err);

% %-----
% % f0_npsi and g0_npsi
% %-----

f1_psi = strict_taylor(cf(:,2),q_tilde0,psi_tilde0,1)+err;
f2_psi = strict_taylor(cf(:,4),q_tilde0,psi_tilde0,1)+err;

% derivative with respect to psi (not tilde psi)
f0_psi = f1_psi + f2_psi/omega0^2;

g0_psi = strict_taylor(cf(:,6),q_tilde0,psi_tilde0,1)+err;

%-----
% % interpolation error of interpolant derivative
%-----

% To bound the interpoaltion error coming from approximating the derivative
% of f
% with the derivative of the interpolating polynomial p, we need a bound on
% the derivative of f on a stadium. We use Cauchy's integral formula to get that
% bound.

```

```

% We already have a bound on f on a stadium of radius rho_q. We choose-
rho_q_sm and
% rho_psi_sm so that the smaller ellipse has a minor axis of half the length of
the larger one.
% Then we use Cauchy's integral formula with f on the larger stadium to get a
bound on the
% derivative of f on the smaller stadium. We then use the radius of the smaller
stadium to get
% the needed interpolation error bounds for the derivative of f.

```

```

% This is a bound on the derivative of f on the smaller stadium
% derived using Cauchy's integral formula
M_Dq = (1/nm(2))*sqrt(d.rho_q^2+1/d.rho_q^2)*d.M_q_n1/min_diff_q^2;
% This is a bound on the length of the smaller stadium
Lq = pie*sqrt(rho_q_sm^2+1/rho_q_sm^2);
% This is a lower bound on the distance between the
% stadium and the line on which we interpolate
Ddq = (rho_q_sm+1/rho_q_sm)/2-1;
eta_q = log(rho_q_sm);
err_Dq = M_Dq*Lq/(pie*Ddq*sinh(eta_q*(d.N_q_n1+1)));

```

```

% Next we find the 2-d interpolation error for the derivative of f with
% respect to q

```

```

Drhoq = (d.rho_q+1/d.rho_q)/2-1;
fc = (d.N_q_n1+1)*(d.N_q_n1+3)/Drhoq+1/Drhoq^2;
err_Dqf = fc*((pie*sqrt(d.rho_q^2+1/d.rho_q^2))*d.M_q_n1/(2*pie*sinh(log(d.rho_q)*(d.N_q_n1+1))));

```

```

% interpolation error
lam_n1_psi = (2/pie)*log(d.N_psi_n1)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_psi_n1));
err_k_q = err_Dq+lam_n1_psi*err_Dqf;
err_k_q = nm(-err_k_q,err_k_q)+1i*nm(-err_k_q,err_k_q);

```

```

%-----
% auxiliary for taking derivative in k
%-----

```

```

kappa = kappa_of_k(k);
K = elliptic_integral(k,1);
E = elliptic_integral(k,2);

```



```
R = elliptic_integral(sqrt(1-k^2),1);
T = elliptic_integral(sqrt(1-k^2),2);
```

```
% Derivative of complete elliptic integral of the first kind
%  $\frac{\partial}{\partial k} K(k)$  K_k = -K/k+E/(k*(1-k^2));
```

```
sqk = sqrt(1-k^2);R_k = -k*(-R/sqk+T/(k^2*sqk))/sqk;
```

```
q_k = (-pie*R_k/K + pie*K_k*R/K^2)*exp(-pie*R/K);
```

```
qtilde_q = 2/(d.b_q-d.a_q);
```

```
omega = pie/kappa;
```

```
kap_k = pie*sqrt((E*(2*k^4 - 2*k^2 + 2) + K*(-k^2 + 2)*(k^2 - 1))/(E*(-
2*k^6 + 3*k^4 + 3*k^2 - 2) + K*(k^6 + k^4 - 4*k^2 + 2)))*(E*(-2*k^6 +
3*k^4 + 3*k^2 - 2) + K*(k^6 + k^4 - 4*k^2 + 2))*((E*(2*k^4 - 2*k^2 +
2) + K*(-k^2 + 2)*(k^2 - 1))*(-E*(-12*k^5 + 12*k^3 + 6*k) - K*(6*k^5 +
4*k^3 - 8*k) - (E/(k*(-k^2 + 1)) - K/k)*(k^6 + k^4 - 4*k^2 + 2) - (E -
K)*(-2*k^6 + 3*k^4 + 3*k^2 - 2)/k)/(2*(E*(-2*k^6 + 3*k^4 + 3*k^2 - 2)
+ K*(k^6 + k^4 - 4*k^2 + 2))^2) + (E*(8*k^3 - 4*k) + 2*K*k*(-k^2 + 2)
- 2*K*k*(k^2 - 1) + (-k^2 + 2)*(k^2 - 1)*(E/(k*(-k^2 + 1)) - K/k) + (E -
K)*(2*k^4 - 2*k^2 + 2)/k)/(2*(E*(-2*k^6 + 3*k^4 + 3*k^2 - 2) + K*(k^6
+ k^4 - 4*k^2 + 2)))/(K*(E*(2*k^4 - 2*k^2 + 2) + K*(-k^2 + 2)*(k^2 -
1))) - pie*sqrt((E*(2*k^4 - 2*k^2 + 2) + K*(-k^2 + 2)*(k^2 - 1))/(E*(-2*k^6
+ 3*k^4 + 3*k^2 - 2) + K*(k^6 + k^4 - 4*k^2 + 2)))*(E/(k*(-k^2 + 1)) -
K/k)/K^2 ;kappa_k = sqrt(nm(7)/20)*kap_k;omega_k = -pi*kappa_k/kappa^2;
```

$f_{0\psi k}$ and $g_{\psi k}$

```
f1_psi_tilde_q = strict_taylor(cf(:,,2),q_tilde0,psi_tilde0,3)+err_k_q;f2_psi_tilde_q
= strict_taylor(cf(:,,4),q_tilde0,psi_tilde0,3)+err_k_q;
```

```
f1_psi_k = q_k*qtilde_q*f1_psi_tilde_q;f2_psi_k = q_k*qtilde_q*f2_psi_tilde_q;
```

```
f0_psi_k = f1_psi_k+f2_psi_k/omega^2-2*f2_psi*omega_k/omega^3;
```

```
g_psi_tilde_q = strict_taylor(cf(:,,6),q_tilde0,psi_tilde0,3)+err_k_q;g0_psi_k = q_k*qtilde_q*g_psi_tilde_q;
```

bound on $f_{\psi\psi k}$ and $g_{\psi\psi k}$

```

psi_tildeL = 2*psiL-1;temp = nm(psi_tildeL,psi_tilde0);tilde_psi = linspace(inf(temp),sup(temp),1000);

f1_psi_psi_tilde_q = strict_taylor(cf(:,7),q_tilde0,tilde_psi,3)+err_k_q;f2_psi_psi_tilde_q-
= strict_taylor(cf(:,8),q_tilde0,tilde_psi,3)+err_k_q;f2_psi_psi = strict_taylor(cf(:,8),q_tilde0,tilde_psi,1)+err;

f1_psi_psi_k = q_k*qtilde_q*f1_psi_psi_tilde_q;f2_psi_psi_k = q_k*qtilde_q*f2_psi_psi_tilde_q;

f_psi_psi_k = f1_psi_psi_k+f2_psi_psi_k/omega^2-2*f2_psi_psi*omega_k/omega^3;

g_psi_psi_tilde_q = strict_taylor(cf(:,9),q_tilde0,tilde_psi,3)+err_k_q;g_psi_psi_k
= q_k*qtilde_q*g_psi_psi_tilde_q;

M_f_psi_psi_k = max(sup(abs(f_psi_psi_k)));

M_g_psi_psi_k = max(sup(abs(g_psi_psi_k)));

----- bound on  $f_{\psi\psi}$  and  $g_{\psi\psi}$  -----
-----

f1_psi_psi = strict_taylor(cf(:,7),q_tilde0,tilde_psi,1)+err;f2_psi_psi = strict_taylor(cf(:,8),q_tilde0,tilde_psi,1)+
= f1_psi_psi+f2_psi_psi/omega^2;

g_psi_psi = strict_taylor(cf(:,9),q_tilde0,tilde_psi,1)+err;

M_f_psi_psi = max(sup(abs(f_psi_psi)));

M_g_psi_psi = max(sup(abs(g_psi_psi)));

----- combine -----
-----

wid = psi0-psiL;wid = nm(-wid,wid);

t1 = f0_psi_k+M_f_psi_psi_k*wid/2;t2 = g0_psi+ M_g_psi_psi*wid/2;t3 = f0_psi
+ M_f_psi_psi*wid/2;t4 = g0_psi + M_g_psi_psi*wid/2;t5 = g0_psi_k + M_g_psi_psi_k*wid/2;

out = t1/t2 - (t3/t4)*(t5/t4);

if sup(real(out)) >= 0    error('failed to verify the derivative is negative') end

```

3.44 theta_vec.m

```

function out = theta_vec(q,psi,x,m,ntilde)
% out = theta_vec(q,psi,x,m,ntilde)
%
% Returns an interval enclosure of f(x) and its first m derivatives where
% f(x) = v(pie(x + n)/2+i*pi*omega_prime*(1+psi)/(2*omega))
% and v(z) is the first Jacobi Theta function with nome q.
%
% The input should satisfy -1 <= x <= 1, 0 <= psi <= 1, 0<q<1, ntilde = 0
or 1, m >= 0

```

The first Jacobi Theta function is given by the series,

$$\vartheta_1(z) = 2 \sum_{n=0}^{\infty} (-1)^n q^{(n+1/2)^2} \sin((2n+1)z).$$

From

$$\vartheta_1(z) = 2 \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} \sin((2n-1)z),$$

we find that

$$\begin{aligned} f(x) &:= \vartheta_1 \left(\frac{\pi}{2\omega} (\omega x + i\omega' + \tilde{n}\omega + i\psi\omega') \right) \\ &= -i \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n-1)} \right), \end{aligned}$$

where $\hat{v} := e^{i\pi(x+\tilde{n})/2} q^{(1+\psi)/2}$. Hence

$$\frac{\partial^m}{\partial x^m} f(x) = -i(i\pi/2)^m \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} (2n-1)^m \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n-1)} \right).$$

We find that

$$\begin{aligned}
Err &:= \left| -i(i\pi/2)^m \sum_{n=N+1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} (2n-1)^m \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n+1)} \right) \right| \\
&\leq 2q^{1/4}(\pi/2)^m \sum_{n=N+1}^{\infty} q^{n^2/2} \left((2n-1)^m q^{n^2/2-n-(2n-1)(1+\psi)/2} \right) \\
&\leq 2q^{1/4}(\pi/2)^m \sum_{n=N+1}^{\infty} q^{n^2/2} \\
&\leq 2q^{1/4}(\pi/2)^m q^{(N+1)^2/2} \frac{1}{1-q},
\end{aligned}$$

as long as we take N large enough so that $g(x) := \left((2x-1)^m q^{n^2/2-x-(2x-1)(1+\psi)/2} \right)$ satisfies $g(N) < 1$, $g(x) < 0$ whenever $x > N$.

%%

%% Error checking

%

% I tested against theta1_m for accuracy, which was tested against Maple and Mathematica. I

% found that they agreed. In testing, I found that this function has error radius about twice that of

% theta1_m for $m = 4$, but that it computed much faster. For example, it took nearly 2 minutes to

% evaluate theta1_m on 200 x points but only took about 0.2 seconds with this vectorized version.

%

% Below is code used for testing:

% x = nm(linspace(-1,1,30));

% q = nm('0.5');

% psi = nm(linspace(0,1,20));

% xicon = 0;

% xicon_der = 0;

%

% tic

% val = integrands(x,q,psi,xicon,xicon_der);

% toc

%

% pie = nm('pi');

% con = pie/2;

% diff = 0;

```

% for j = 1:length(x)
%   for k = 1:length(psi)
%
%       z = pie*(x(j) + 1)/2-log(q)*1i*(1+psi(k))/(2);
%       temp = theta1_m(z,q,4,1e-17);
%       for ind = 1:5
%           diff = max(diff,sup(temp(ind)*con^(ind-1)-val(j,k,ind)));
%       end
%   end
% end
%
% disp(diff);
% 5.975892748039020e-11 + 5.958839722380777e-11i

%%

% error target
tol = 1e-17;

%
% constants
%

psi0 = max(sup(psi));
pie = nm('pi');
one = nm(1);
req_1 = sup(real(-2*m/log(q)));
c1 = nm(2+psi0);
c2 = nm(2*q^(one/4)*(pie/2)^m/(1-q));
qsqrt = q^(one/2);
half = one/2;

%
% find N large enough that truncation error is less than tol
%

err = tol + 1;
N = 0;
maxit = 1000;
while err > tol

```

```

N = N + 1;
if N > maxit
    error('maximum iterations exceeded');
end

% check that derivative of  $f(x) := q^{x^2/2-x-(2x-1)(1+\psi)/2}(2x-1)^m$  is
decreasing for  $x \geq N$ .
if inf(real((N-c1)*(2*N-1))) <= req_1
    continue
end

% check that  $f(x) := q^{N^2/2-N-(2N-1)(1+\psi)/2}(2N-1)^m \leq 1$ 
if sup(real(q^(N^2/2-N-(2*N-1)*(1+psi0)/2)*(2*N-1)^m)) > 1
    continue
end

% turncation error
err = sup(real(c2*qsqrt^((N+1)^2)));

end

%
% evaluate the partial sum
%

% initialize vectors
out = nm(zeros(length(x),length(psi),m+1));
temp = out;

% rearrange dimensions if needed
sx = size(x,1);
if sx == 1
    x = x.';
end

sx = size(psi,1);
if sx > 1
    psi = psi.';
end

%

```

```

% add partial sum
%

vhat = exp(1i*pi*(x+ntilde)/2)*q.^((1+psi)/2);

for n = 1:N

    vtemp = vhat.^(2*n-1);
    for k = 0:m
        temp(:,k+1) = (2*n-1)^k*(vtemp - (-1)^k*vtemp.^(-1));
    end

    out = out + (-1)^(n+1)*q^((n-half)^2)*temp;

end

% complete differentiation rule
for j = 0:m
    out(:,j+1) = out(:,j+1)*(1i*pi/2)^j;
end

% multiply by -1i coming from sin definition and add error interval
out = -1i*out + nm(-err,err)+1i*nm(-err,err);

```

3.45 upper_pinpoint.m

```

curr_dir = local_startup; clc;
intvalinit('DisplayMidRad');

% retrieve bounds on function

file_name = 'interp2d_490_to_538';

```

```

ld = retrieve_it(curr_dir,'interval_arithmetic',file_name,'data_final');
d = ld.var;

% constants
pie = nm('pi');

ntilde = 0;
% initialize function
fun = (q,psi)(integrand_numer(d.N_x_n0,d.err_x_n0,q,psi,ntilde));

min_wid = 1e-8;

kstart = nm('0.9999989');
psiL = inf(nm('0.7'));
psiR = sup(nm('0.8'));

% confirm stability above
for j = 18:53

    k = nm(mid(kstart-nm(1)/2^j));

    kappa = kappa_of_k(k);
    elipk = elliptic_integral(k,1);
    elipk2 = elliptic_integral(sqrt(1-k^2),1);
    omega = pie/kappa;
    X = 2*pie/kappa;
    q = exp(-pie*elipk2/elipk);

    if (sup(q) > inf(d.b_q))——(inf(q)< sup(d.a_q))
        error('q out of range');
    end

    psiv = linspace(psiL,psiR,200);
    out = fun(q,psiv);
    f = out(:,1)+out(:,3)/omega^2;
    f = sup(imag(f));

    if sum(any(f<0))>0
        kstart = k;
        ind = find(f < 0);
    end
end

```



```

        psiL = psiv(max(ind(1)-1,1));
        psiR = psiv(min(ind(end)+1,length(psiv)));
    end

end

kstart = sup(kstart);

% confirm stability below
for j = 30:1:53

    k = nm(mid(kstart-nm(1)/2^j));

    kappa = kappa_of_k(k);
    elipk = elliptic_integral(k,1);
    elipk2 = elliptic_integral(sqrt(1-k^2),1);
    omega = pie/kappa;
    X = 2*pie/kappa;
    q = exp(-pie*elipk2/elipk);

    if (sup(q) > inf(d.b_q))——(inf(q)< sup(d.a_q))
        error('q out of range');
    end

    psiL = inf(nm('0.7'));
    psiR = sup(nm('0.8'));

    psi_keep_going = 1;
    while psi_keep_going == 1

        psiv = linspace(psiL,psiR,200);

        out = fun(q,psiv);
        f_psi = out(:,2)+out(:,4)/omega^2;

        indL = find(sup(imag(f_psi)) < 0);
        indR = find(inf(imag(f_psi)) > 0);

        disp(' ')
        psiL = psiv(indL(end));

```

```

psiR = psiv(indR(1));

temp = fun(q,nm(psiL,psiR));

test = temp(:,1)+temp(:,3)/omega^2;

if inf(imag(test)) > 0
    psi_keep_going = 0;
elseif sup(imag(test)) < 0
    k_stab = nm(mid(kstart-nm(1)/2^(j-1)));
    return
elseif psiR-psiL < min_wid
    k_stab = nm(mid(kstart-nm(1)/2^(j-1)));
    return
end

end
end

kstab = inf(k_stab);

%   psiL =   0.759188696240224
%   psiR =   0.759188699683576

%   verified stable at:      k = 0.999998385205026, X = 26.05736207014433
%   verified unstable at :   k = 0.999998385263233, X = 26.05742300506267

```

3.46 verify_instability_lower.m

```

function [k_left,k_right,min_quot] = verify_instability_lower(d,k)
%-----
% verify instability
%-----

% interval pi
pie = nm('pi');

```

```

% interpolation error
err = d.err_q_n1;

kappa = kappa_of_k(k);
omega = pie./nm(kappa(1:end-1),kappa(2:end));

% make intervals between k values
k = nm(k(1:end-1),k(2:end));
% constants for transforming coordinats
c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
% get q values and omega values
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
q = exp(-pie*elipk2./elipk);
% make transformation in q to qtilde in [-1,1]
q_tilde = c1_q*(q-c2_q);
psi_tilde = nm(1);

% make sure q is within range of data in structure d
if min(inf(q)) < sup(d.q_min)
    error('q out of range');
end

% make sure q is within range of data in structure d
if max(sup(q)) > d.q_max
    error('q out of range');
end

% get interpolated values and add error
f1_psi = (cf_eval(d.cfn1(:,2),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
f2_psi = (cf_eval(d.cfn1(:,4),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);
g_psi = (cf_eval(d.cfn1(:,6),q_tilde,psi_tilde))+nm(-err,err)+1i*nm(-err,err);

% % process data to get function values

omega = omega.';
quot =real((f1_psi./g_psi) + (f2_psi./g_psi)./omega.^2);

```

```

% find where instability is verified
ind = find(inf(quot)>0);

% throw error if no instability verified
if isempty(ind)
    error('failed to verify');
end

% make sure there are no gaps on what we report
% as having been verified as unstable
left = ind(1);
right = ind(1);
for j = 1:length(ind)-1
    if ind(j+1) == ind(j)+1
        right = ind(j+1);
    else
        break
    end
end

% get left and right k values that were verified.
k_left = mid(k(left));
k_right = mid(k(right));

% find the minimum of the quotient for verified values
min_quot = min(inf(quot(left:right)));

```

3.47 verify_instability_upper.m

```

function verify_instability_upper(d, kleft, kright, psi_left, psi_right, ints_y)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% form k interval
k = [nm(kleft), nm(kright)];

```

```

% coefficients
cf = d.cf10;
% constants
pie = nm('pi');
% interpolation error
lam_10_q = (2/pie)*log(d.N_q-10)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q-10));
err = d.err_q-10+lam_10_q*d.err_psi-10;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% constants for transformation in q
c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
% kappa
kappa = kappa_of_k(k);
% period
X = 2*pie./kappa;
% fprintf('\nX: ')
% sup(X)
% elliptic integrals
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
% q
q = exp(-pie*elipk2./elipk);
% omega
omega = pie./kappa;
% get theta values for q
q_tilde = c1_q*(q-c2_q);
theta_q = flplr(acos(q_tilde));
c1_psi = nm(2);
c2_psi = nm(1)/2;

% check user input is correct
if inf(q(1)) < inf(d.a_q)
    error('k out of range');
end

if sup(q(2)) > sup(d.b_q)
    error('k out of range');
end

lty = acos((2*psi_left-1));
rty = acos((2*psi_right-1));

```

```

ltx = inf(theta_q(1));
rtx = sup(theta_q(2));

% interpolate
c0 = cheby_eval(cf(:,1),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
c1 = cheby_eval(cf(:,2),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
c2 = cheby_eval(cf(:,3),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
c3 = cheby_eval(cf(:,4),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
h0 = cheby_eval(cf(:,5),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
h1 = cheby_eval(cf(:,6),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
h2 = cheby_eval(cf(:,7),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
h3 = cheby_eval(cf(:,8),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
h4 = cheby_eval(cf(:,9),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);
h5 = cheby_eval(cf(:,10),ltx,rtx,lty,rty,0,ints_y)+nm(-err,err)+1i*nm(-err,err);

ty = linspace(lty,rty,ints_y+1);
psi_tilde = cos(nm(ty));
psi = psi_tilde/c1_psi+c2_psi;

% 1i*xi - 1i*(Floquet parameter)
xicon = 1i*xi_q_psi(nm(q(1),q(2)),psi,0);

% sub functions
f1 = c0 + c1.*xicon.^1 + c2.*xicon.^2 + c3.*xicon.^3;
f2 = h0 + h1.*xicon.^1 + h2.*xicon.^2 + h3.*xicon.^3 + ...
      h4.*xicon.^4 + h5.*xicon.^5;

%-----
% check that the middle of numerator is positive
%-----

% numerator of lambda_1
numer = f1+f2/nm(omega(1),omega(2))^2;

numer = sup(imag(numer));

```

```

% check for NaNs
if sum(any(isnan(number))) > 0
    error('NaN present');
end

% instability test: WTS  $f_1 + f_2/\omega^2 < 0$ 
if sum(any(number < 0)) == 0
    error('failed to verify instability');
end

```

3.48 verify_stability_n0_strict.m

```

function verify_stability_n0_strict(d, kleft, kright, ints_yL, ints_yR, ...
    psi_L, psi_M, psi_R, psi_R2, psi_R3, stats)

% number of intervals in q (keep fixed at 1)
ints_x = 1;
% ensure that psi_L, psi_M, and psi_R are intervals
psi_L = nm(psi_L);
psi_M = nm(psi_M);
psi_R = nm(psi_R);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% form k interval
k = [nm(kleft), nm(kright)];
% coefficients
cf = d.cf10;
% constants
pie = nm('pi');
% interpolation error
lam_10_q = (2/pie)*log(d.N_q_10)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_10));
err = d.err_q_10+lam_10_q*d.err_psi_10;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% constants for transformation in q
c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
% kappa
kappa = kappa_of_k(k);

```

```

% period
X = 2*pie./kappa;
% elliptic integrals
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
% q
q = exp(-pie*elipk2./elipk);
% omega
omega = pie./kappa;
% get theta values for q
q_tilde = c1_q*(q-c2_q);
theta_q = fliplr(acos(q_tilde));
c1_psi = nm(2);
c2_psi = nm(1)/2;
ltx = inf(theta_q(1));
rtx = sup(theta_q(2));

% check user input is correct
if inf(q(1)) < inf(d.a_q)
    error('k out of range');
end

if sup(q(2)) > sup(d.b_q)
    error('k out of range');
end

psi_tilde_L = 2*psi_L-1;
psi_tilde_M = 2*psi_M-1;
rty = inf(acos(psi_tilde_L));
lty = sup(acos(psi_tilde_M));

% interpolate
c0 = cheby_taylor(cf(:,1),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
c1 = cheby_taylor(cf(:,2),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
c2 = cheby_taylor(cf(:,3),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
c3 = cheby_taylor(cf(:,4),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h0 = cheby_taylor(cf(:,5),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h1 = cheby_taylor(cf(:,6),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h2 = cheby_taylor(cf(:,7),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h3 = cheby_taylor(cf(:,8),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h4 = cheby_taylor(cf(:,9),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h5 = cheby_taylor(cf(:,10),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-

```



```

err,err);

ty = linspace(lty,rty,ints_yL+1);
psi_tilde = cos(nm(ty));
psi = psi_tilde/c1_psi+c2_psi;

% li*xi - li*(Floquet parameter)
xicon = li*xi_q_psi(nm(q(1),q(2)),psi,0);
xicon = nm(xicon(1:end-1),xicon(2:end));
xicon = repmat(xicon,ints_x,1);

% sub functions
f1 = c0 + c1.*xicon.^1+ c2.*xicon.^2 + c3.*xicon.^3;
f2 = h0 + h1.*xicon.^1+ h2.*xicon.^2 + h3.*xicon.^3 + ...
    h4.*xicon.^4 + h5.*xicon.^5;

%-----
% check that the middle of numerator is positive
%-----

% numerator of lambda_1
numer = f1+f2/nm(omega(1),omega(2))^2;

if strcmp(stats,'on')
    figure
    hold on
    psi_tildep = cos(ty(1:end-1));
    plot((psi_tildep+1)/2,inf(imag(numer)),'-r','LineWidth',2);
    plot((psi_tildep+1)/2,sup(imag(numer)),'-b','LineWidth',2);
    h = xlabel('psi');
    set(h,'FontSize',22);
    h = ylabel('f');
    set(h,'FontSize',22);
    h = gca;
    set(h,'FontSize',22);
end

numer = inf(imag(numer));

```

```

% check for NaNs
if sum(any(isnan(number))) > 0
    error('NaN present');
end

% stability test: WTS  $f_1 + f_2/\omega^2 > 0$ 
if sum(any(number <=0)) > 0
    error('failed to verify stability on middle part of numerator');
end

%-----
% check numerator on left
%-----

psi_tilde_L = 2*psi_L-1;
rty = sup(pie);
lty = sup(acos(psi_tilde_L));

% cut-off left psi value
left_psi = cos(nm(lty))/c1_psi+c2_psi;

% interpolate
c0 = cheby_taylor(cf(:,1),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
c1 = cheby_taylor(cf(:,2),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
c2 = cheby_taylor(cf(:,3),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
c3 = cheby_taylor(cf(:,4),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h0 = cheby_taylor(cf(:,5),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h1 = cheby_taylor(cf(:,6),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h2 = cheby_taylor(cf(:,7),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h3 = cheby_taylor(cf(:,8),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h4 = cheby_taylor(cf(:,9),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
h5 = cheby_taylor(cf(:,10),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);

if sum(any(isnan(c0))) > 0
    error('NaN present');
end

```

```

if sum(any(isnan(c1))) > 0
    error('NaN present');
end
if sum(any(isnan(c2))) > 0
    error('NaN present');
end
if sum(any(isnan(c3))) > 0
    error('NaN present');
end
if sum(any(isnan(h0))) > 0
    error('NaN present');
end
if sum(any(isnan(h1))) > 0
    error('NaN present');
end
if sum(any(isnan(h2))) > 0
    error('NaN present');
end
if sum(any(isnan(h3))) > 0
    error('NaN present');
end
if sum(any(isnan(h4))) > 0
    error('NaN present');
end
if sum(any(isnan(h5))) > 0
    error('NaN present');
end

% get polynomial coefficient extremes
omeg = nm(omega(1),omega(2));
p0 = max(max(sup(abs(c0+h0./omeg.^2))));
p1 = max(max(sup(abs(c1+h1./omeg.^2))));
p2 = max(max(sup(abs(c2+h2./omeg.^2))));
p3 = max(max(sup(abs(c3+h3./omeg.^2))));
p4 = max(max(sup(abs(h4./omeg.^2))));
p5 = min(min(inf(abs(h5./omeg.^2))));

% form bound on roots
R = nm(1)+(1/nm(p5))*nm(max([p0,p1,p2,p3,p4]));
R = sup(R);
% determine  $\xi$  at left_psi
xicon = abs(xi_q_psi(nm(q(1),q(2)),left_psi,0));

```

```

% check if  $-\xi < R$ 
if inf(real(xicon)) <= R
    error('failed to verify on left')
end

%-----
% middle region
%-----

lam_q = (2/pie)*log(d.N_q_n0)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n0));
err = d.err_q_n0+lam_q*d.err_psi_n0;

% left theta region corresponding to right x region
psi_tilde_R = 4*psi_R-3;
psi_tilde_M = 4*psi_M-3;
rty = inf(acos(psi_tilde_M));
lty = sup(acos(psi_tilde_R));

f1 = cheby_taylor(d.cfn0(:,1),ltx,rtx,lty,rty,ints_x,ints_yR)+nm(-err,err)+1i*nm(-err,err);
f2 = cheby_taylor(d.cfn0(:,3),ltx,rtx,lty,rty,ints_x,ints_yR)+nm(-err,err)+1i*nm(-err,err);
numer = f1+ f2/nm(omega(1),omega(2))^2;

if strcmp(stats,'on')
    hold on
    typ = linspace(lty,rty,ints_yR+1);
    psi_tildep = cos(typ(1:end-1));
    plot((psi_tildep+3)/4,inf(imag(numer)),'-m','LineWidth',2);
    plot((psi_tildep+3)/4,sup(imag(numer)),'-c','LineWidth',2);
    h = xlabel('psi');
    set(h,'FontSize',22);
    h = ylabel('f');
    set(h,'FontSize',22);
    h = gca;
    set(h,'FontSize',22);
end

numer = inf(imag(numer));

```

```

% check for NaNs
if sum(any(isnan(number))) > 0
    error('NaN present');
end

% stability test: WTS  $f_1 + f_2/\omega^2 > 0$ 
if sum(any(number <=0)) > 0
    error('failed to verify stability on middle part of numerator');
end

%-----
% derivatives on right
%-----

lam_q = (2/pie)*log(d.N_q_n0)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n0));
err = d.err_q_n0+lam_q*d.err_psi_n0;

psi_tilde_R = 4*psi_R3-3;
lty = 0;
rty = sup(acos(psi_tilde_R));

% get interpolated polynomials
f1_psi = cheby_taylor(d.cfn0(:,2),ltx,ltx,lty,rty,ints_x,ints_yL)+nm(-err,err)+1i*nm(-err,err);
f2_psi = cheby_taylor(d.cfn0(:,4),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+1i*nm(-err,err);

%-----
% derivative of numerator on right
%-----

numer_psi = f1_psi+f2_psi/nm(omega(1),omega(2))^2;

if strcmp(stats,'on')
    hold on
    typ = linspace(lty,rty,ints_yL+1);
    psi_tildep = cos(typ(1:end-1));

```

```

plot((psi_tilde+3)/4,sup(imag(numer_psi)),'-r')
plot((psi_tilde+3)/4,inf(imag(numer_psi)),'-g')
h = xlabel('psi');
set(h,'FontSize',22);
h = ylabel('f');
set(h,'FontSize',22);
h = gca;
set(h,'FontSize',22);
end

numer_psi = sup(imag(numer_psi));

if sum(any(isnan(numer_psi))) > 0
    error('NaN present');
end

% stability test: equivlanet to showing f1+f2/omega^2 < 0
if sum(any(numer_psi >=0))
    error('failed to verify stability on right part of numerator');
end

%-----
% right middle part
%-----

lam_q = (2/pie)*log(d.N_q_n0)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n0));
err = d.err_q_n0+lam_q*d.err_psi_n0;

psi_tilde_R = 4*psi_R3-3;
psi_tilde_L = 4*psi_R2-3;
lty = sup(acos(psi_tilde_L));
rty = sup(acos(psi_tilde_R));

% get interpolated polynomials
f1 = cheby_taylor(d.cfn0(:,1),ltx,ltx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);
f2 = cheby_taylor(d.cfn0(:,3),ltx,rtx,lty,rty,ints_x,ints_yL)+nm(-err,err)+li*nm(-err,err);

```

```

numer = f1+f2/nm(omega(1),omega(2))^2;

numer = inf(imag(numer));

if sum(any(isnan(numer))) > 0
    error('NaN present');
end

% stability test: equivlanet to showing f1+f2/omega^2 < 0
if sum(any(numer <=0))
    error('failed to verify stability on right part of numerator');
end

```

3.49 verify_stability_n1.m

```

function verify_stability_n1(d, kleft, kright, ints_x,ints_y)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% form k interval
k = [nm(kleft), nm(kright)];
% coefficients
cf = d.cfn1;
pie = nm('pi');
% interpolation error
lam_n1_q = (2/pie)*log(d.N_q_n1)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n1));
err = d.err_q_n1+lam_n1_q*d.err_psi_n1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% constants for transformation in q
c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
% kappa
kappa = kappa_of_k(k);
% elliptic integrals
% X = 2*pie./kappa;
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
% q

```

```

q = exp(-pie*elipk2./elipk);
% omega
omega = pie./kappa;
% get theta values for q
q_tilde = c1_q*(q-c2_q);
theta_q = fliplr(acos(q_tilde));

% check user input is correct
if inf(q(1)) < inf(d.a_q)
    error('k out of range');
end

if sup(q(2)) > sup(d.b_q)
    error('k out of range');
end

% specify middle of three theta regions
% lty = inf(acos(psi_tilde_L))
lty = inf(2*pie/10);
rty = sup((9*pie)/10);
ltx = inf(theta_q(1));
rtx = sup(theta_q(2));

% evalute interpolating polynomials
f1 = cheby_taylor(cf(:,1),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+li*nm(-err,err);
f2 = cheby_taylor(cf(:,3),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+li*nm(-err,err);
g = cheby_taylor(cf(:,5),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+li*nm(-err,err);

%-----
% check that the middle of numerator is positive
%-----

f1 = imag(f1);
f2 = imag(f2);

numer = f1+f2/nm(omega(1),omega(2))^2;

if sum(any(isnan(numer))) > 0

```



```

        error('NaN present');
    end

% stability test: equivalent to showing f1 + f2/omega^2 > 0
    if sum(any(inf(number)<=0)) > 0
        error('failed to verify stability on middle part of numerator');
    end

%-----
% check that the middle of denominator is positive
%-----

g = imag(g);

    if sum(any(isnan(g))) > 0
        error('NaN present');
    end

% stability test
    if sum(any(sup(g) >=0)) > 0
        error('failed to verify stability on middle part of numerator');
    end

%-----
% derivatives on left
%-----

% specify theta region on right, corresponding to region in x on left
rty = pie;
lty = inf(9*pie/10);

% evaluate interpolation polynomials
f1_psi = cheby_taylor(cf(:,2),ltx,ltx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-err,err);
f2_psi = cheby_taylor(cf(:,4),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-err,err);
g_psi = cheby_taylor(cf(:,6),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-err,err);

```

```

%-----
% derivative of numerator on left
%-----

f1_psi = imag(f1_psi);
f2_psi = imag(f2_psi);

% error checking
if sum(any(inf(f2_psi) <= 0)) > 0
    error('f2_psi has non-positive part')
end

quot = inf(f1_psi./f2_psi+1/omega(2)^2);

if sum(any(isnan(quot))) > 0
    error('NaN present');
end

% stability test: equivalent to showing f1+f2/omega^2 > 0
if any(quot <=0)
    error('failed to verify stability on left part of numerator');
end

%-----
% derivative of denominator on left
%-----

g_psi = imag(g_psi);

% error checking
if sum(any(isnan(g_psi))) > 0
    error('NaN present');
end

% stability test

```

```

if any( sup(g_psi) >=0)
    error('failed to verify stability on left part of denominator');
end

%-----
% derivatives on right
%-----

% left theta region corresponding to right x region
rty =sup(2*pie/10);
lty = 0;

% get interpolated polynomials
f1_psi = cheby_taylor(cf(:,2),ltx,ltx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);
f2_psi = cheby_taylor(cf(:,4),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);

%-----
% derivative of numerator on right
%-----

f1_psi = imag(f1_psi);
f2_psi = imag(f2_psi);

% error checking
if sum(any(sup(f2_psi) >= 0)) > 0
    error('f2_psi has non-negative part')
end

quot = inf(f1_psi./f2_psi+1/omega(2)^2);

if sum(any(isnan(quot))) > 0
    error('NaN present');
end

% stability test: equivlanet to showing f1+f2/omega^2 < 0

```

```

if any(quot <=0)
    this = quot(find(quot<=0));
    this(1:min(10,length(this)))
    error('failed to verify stability on right part of numerator');
end

%-----
% derivative of denominator on right
%-----

% left theta region corresponding to right x region
rty =sup(2*pie/10);
lty = 0;

g_psi = cheby_taylor(cf(:,6),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);
g_psi = imag(g_psi);

% error checking
if sum(any(isnan(g_psi))) > 0
    error('NaN present');
end

% stability test
if sum(any( sup(g_psi)) <=0)
    error('failed to verify stability on right part of denominator');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

3.50 verify_stability_n1_strict.m

```

function verify_stability_n1_strict(d, kleft, kright, ints_x,ints_y,psiL)

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% form k interval
k = [nm(kleft), nm(kright)];
% coefficients
cf = d.cfn1;
pie = nm('pi');
% interpolation error
lam_n1_q = (2/pie)*log(d.N_q_n1)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n1));
err = d.err_q_n1+lam_n1_q*d.err_psi_n1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% constants for transformation in q
c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
% kappa
kappa = kappa_of_k(k);
% elliptic integrals
% X = 2*pi./kappa
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
% q
% q = exp(-pie*elipk2./elipk)
% omega
omega = pie./kappa;
% get theta values for q
q_tilde = c1_q*(q-c2_q);
theta_q = fliplr(acos(q_tilde));

% check user input is correct
if inf(q(1)) < inf(d.a_q)
    error('k out of range');
end

if sup(q(2)) > sup(d.b_q)
    error('k out of range');
end

% specify middle of three theta regions
% lty = inf(acos(psi_tilde_L))
lty = inf(2*pi/10);
rty = sup((9*pi)/10);

```

```

ltx = inf(theta_q(1));
rtx = sup(theta_q(2));

% evaluate interpolating polynomials
f1 = cheby_taylor(cf(:,1),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-err,err);
f2 = cheby_taylor(cf(:,3),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-err,err);
g = cheby_taylor(cf(:,5),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-err,err);

%-----
% check that the middle of numerator is positive
%-----

f1 = imag(f1);
f2 = imag(f2);

% error checking
if sum(any(inf(f2) <= 0)) > 0
    ind= find(inf(f2)<=0);
    length(ind)
    error('f2 has non-positive part')
end

quot = inf(f1./f2+1/omega(2)^2);

if sum(any(isnan(quot))) > 0
    error('NaN present');
end

% stability test: equivalent to showing f1 + f2/omega^2 > 0
if sum(any(quot <=0)) > 0
    error('failed to verify stability on middle part of numerator');
end

%-----
% check that the middle of denominator is positive
%-----

```

```

g = imag(g);

if sum(any(isnan(g))) > 0
    error('NaN present');
end

% stability test
if sum(any(sup(g) >=0)) > 0
    error('failed to verify stability on middle part of numerator');
end

%-----
% derivatives on left
%-----

% specify theta region on right, corresponding to region in x on left
rty = pie;
lty = inf(9*pie/10);

% evaluate interpolation polynomials
f1_psi = cheby_taylor(cf(:,2),ltx,ltx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);
f2_psi = cheby_taylor(cf(:,4),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);
g_psi = cheby_taylor(cf(:,6),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);

%-----
% derivative of numerator on left
%-----

f1_psi = imag(f1_psi);
f2_psi = imag(f2_psi);

% error checking
if sum(any(inf(f2_psi) <= 0)) > 0
    error('f2_psi has non-positive part')
end

```

```

quot = inf(f1_psi./f2_psi+1/omega(2)^2);

if sum(any(isnan(quot))) > 0
    error('NaN present');
end

% stability test: equivalent to showing f1+f2/omega^2 > 0
if any(quot <=0)
    error('failed to verify stability on left part of numerator');
end

%-----
% derivative of denominator on left
%-----

g_psi = imag(g_psi);

% error checking
if sum(any(isnan(g_psi))) > 0
    error('NaN present');
end

% stability test
if any( sup(g_psi) >=0)
    error('failed to verify stability on left part of denominator');
end

%-----
% derivatives on right
%-----

% left theta regon corresponding to right x region
rty = sup(2*pie/10);
lty = inf(acos(2*psiL-1));

```



```

% get interpolated polynomials
f1_psi = cheby_taylor(cf(:,2),ltx,ltx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);
f2_psi = cheby_taylor(cf(:,4),ltx,ltx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);

%-----
% derivative of numerator on right
%-----

f1_psi = imag(f1_psi);
f2_psi = imag(f2_psi);

% error checking
if sum(any(sup(f2_psi) >= 0)) > 0
    error('f2_psi has non-negative part')
end

quot = inf(f1_psi./f2_psi+1/omega(2)^2);

if sum(any(isnan(quot))) > 0
    error('NaN present');
end

% stability test: equivlanet to showing f1+f2/omega^2 < 0
if any(quot <=0)
    error('failed to verify stability on right part of numerator');
end

%-----
% derivative of denominator on right
%-----

% left theta regon corresponding to right x region
rty =sup(2*pie/10);
lty = 0;

```

```

g_psi = cheby_taylor(cf(:,6),ltx,rtx,lty,rty,ints_x,ints_y)+nm(-err,err)+1i*nm(-
err,err);
g_psi = imag(g_psi);

% error checking
if sum(any(isnan(g_psi))) > 0
    error('NaN present');
end

% stability test
if sum(any( sup(g_psi)) <=0)
    error('failed to verify stability on right part of denominator');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

3.51 verify_stability_single.m

```

function verify_stability_single(d,k,pnts,psi_L,psi_R)

% number of intervals in q (keep fixed at 1)
ints_x = 1;

% ensure that psi_L, psi_M, and psi_R are intervals
psi_L = nm(psi_L);
psi_R = nm(psi_R);

% constants
pie = nm('pi');
% interpolation error

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% constants for transformation in q
c1_q = 2/(d.b_q-d.a_q);
c2_q = (d.a_q+d.b_q)/2;
% kappa
kappa = kappa_of_k(k);
% period
% X = 2*pie./kappa;
% elliptic integrals
elipk = elliptic_integral(k,1);
elipk2 = elliptic_integral(sqrt(1-k.^2),1);
% q
q = exp(-pie*elipk2./elipk);
% omega
omega = pie./kappa;
% get theta values for q
q_tilde = c1_q*(q-c2_q);
theta_q = fliplr(acos(q_tilde));
ltx = inf(theta_q(1));
rtx = sup(theta_q(1));

% check user input is correct
if inf(q) < inf(d.a_q)
    error('k out of range');
end

%-----
% midde middle part
%-----

lam_q = (2/pie)*log(d.N_q_n0)+(2/pie)*(nm('0.6')+log(8/pie)+pie/(72*d.N_q_n0));
err = d.err_q_n0+lam_q*d.err_psi_n0;

psi_tilde_R = 4*psi_R-3;
psi_tilde_L = 4*psi_L-3;
lty = sup(acos(psi_tilde_L));
rty = sup(acos(psi_tilde_R));

% get interpolated polynomials
fl = cheby_taylor(d.cfn0(:,1),ltx,ltx,lty,rty,ints_x,pnts)+nm(-err,err)+1i*nm(-
err,err);

```

```

f2 = cheby_taylor(d.cfn0(:, :, 3), ltx, rtx, lty, rty, ints_x, pnts) + nm(-err, err) + 1i * nm(-err, err);

numer = f1 + f2 / omega ^ 2;

numer = inf(imag(numer));

if sum(any(isnan(numer))) > 0
    error('NaN present');
end

% stability test: equivlanet to showing f1+f2/omega^2 < 0
if sum(any(numer <= 0)) > 0
    min(numer)
    error('failed to verify stability');
end

```

3.52 weierstrass_eta1.m

```

function out = weierstrass_eta1(omega_1, omega_2)
% out = weierstrass_eta1(omega_1, omega_2)
%
% Returns, using interval arithmetic,  $z(\omega_1; g_2, g_3)$  where  $z$  is
% the Weierstrass zeta function.

% nome
q = exp(nm('pi') * nm('1i') * omega_2 / omega_1);

```

```

% check for input error
if real(q) <= nm('0')
    error('q not postive')
end
if real(q) >= nm('1')
    error('q not less than 1');
end
if sup(abs(imag(q))>0)
    error('q not real');
end

% find k so that  $kq^k \downarrow 1$  and  $q(1+q^k) \downarrow 1$ 
k = 1;
proceed = 1;
while proceed == 1
    k = k+1;
    ks = nm(num2str(k));
    if sup(ks*q^ks-nm('1'))<0
        if sup(q*(1+q^ks)-nm('1'))<0
            proceed =0;
        end
    end
end

% find k so that the error bound is less than tol = 1e-16
error_bound = abs((q^ks/(1-q^(nm('2')*ks)))*nm('1')/(nm('1')-q));
while sup(error_bound) > 1e-16
    k = k+1;
    ks = nm(num2str(k));
    error_bound = abs((q^ks/(1-q^(nm('2')*ks)))/(nm('1')-q));
end

% add the first k-1 terms of the sum
sum = nm('0');
for j = 1:k-1
    js = nm(num2str(j));
    sum = sum + js*q^(nm('2')*js)/(nm('1')-q^(nm('2')*js));
end

% add error to sum
sum = sum + hull(-error_bound,error_bound);

```

```

% compute eta_1 with interval sum
pie = nm('pi');
out = pie^nm('2')/(nm('12')*omega_1)-nm('2')*pie^nm('2')*sum/omega_1;

```

3.53 weierstrass_invariants.m

```

function [g2,g3] = weierstrass_invariants(omega_1,omega_2)
% [g2,g3] = weierstrass_invariants(omega_1,omega_2)
%
% Returns, using interval arithmetic, the Weierstrass elliptic function invariants
corresponding to
% half periods omega_1 and omega_2. The half period omega_1 should be
% a positive real and the half period omega_2 should be purely imaginary
% and lie in the upper half of the complex plane.

% period ratio
tau = omega_2/omega_1;

```

```

% Jacobi theta functions
z = nm('0');
theta_3 = theta_func(z,tau);
theta_2 = exp(nm('0.25')*nm('pi')*nm('1i')*tau+nm('pi')*nm('1i')*z)*theta_func(z+nm('0.5')*tau,tau);

% for convenience
pie = nm('pi');

% Weierstrass elliptic function invariants
g2 = pie^nm('4')/(nm('12')*omega_1^nm('4'))*...
    (theta_2^nm('8')-theta_3^nm('4')*theta_2^nm('4')+theta_3^nm('8'));

g3 = (pie^nm('6')/(nm('2')*omega_1^nm('6'))*((nm('8')/nm('27'))*(theta_2^nm('12')+theta_3^nm('12'))...
    -(nm('4')/nm('9'))*(theta_2^nm('4')+theta_3^nm('4'))*theta_2^nm('4')*theta_3^nm('4')));

```

3.54 weierstrass_p.m

```

function out = weierstrass_p(zin,w1,w2)
% out = weierstrass_p(z,w1,w2)
%
% Returns, computed with ineterval arithmetic, the Weierstrass elliptic function
%
% with half periods w1 and w2 evaluated at z. Here w1 is a postive real and
% w2 is purely imaginary lying above the real axis.

% use periodicity of the Weierstrass elliptic function to bring argument
% close to origin where the theta_func program can succesffuly compute
while sup(real(zin)) > sup(real(w1))

```

```

        zin = zin - nm('2')*w1;
    end
    while inf(real(zin)) < -inf(real(w1))
        zin = zin + nm('2')*w1;
    end
    while sup(imag(zin)) > sup(imag(w2))
        zin = zin - nm('2')*w2;
    end
    while inf(imag(zin)) < inf(-imag(w2))
        zin = zin + nm('2')*w2;
    end

    tau = w2/w1;
    z = zin/(nm('2')*w1);

    v01z = theta_func(z+nm('0.5'),tau);
    v11z = exp(nm('0.25')*nm('pi')*nm('1i')*tau+nm('pi')*nm('1i')*(z+nm('0.5')))...
        *theta_func(z+nm('0.5')*tau+nm('0.5'),tau);

    v00zzero = theta_func(nm('0'),tau);
    v10zzero = exp(nm('0.25')*nm('pi')*nm('1i')*tau)*theta_func(nm('0.5')*tau,tau);

    p = nm('pi')^nm('2')*v00zzero^nm('2')*v10zzero^nm('2')*(v01z^nm('2')/v11z^nm('2'))...
        -(nm('pi')^nm('2')/nm('3'))*(v00zzero^nm('4')+v10zzero^nm('4'));

    out = p/(nm('2')*w1)^nm('2');

```

3.55 weierstrass_p_der.m

```

function out = weierstrass_p_der(z,w1,w2,varargin)
% function out = weierstrass_p_der(z,w1,w2,varargin)
%
% Returns, computed with ineterval arithmetic, the derivative of the Weierstrass-
% elliptic function
% with half periods w1 and w2 evaluated at z. Here w1 is a postive real number
% and
% w2 is purely imaginary number lying above the real axis.

```



```

% use periodicity of the Weierstrass elliptic prime function to bring argument
% close to origin where the program can compute more easily
while sup(real(z)) > sup(real(w1))
    z = z - nm('2')*w1;
end
while inf(real(z)) < -inf(real(w1))
    z = z + nm('2')*w1;
end
while sup(imag(z)) > sup(imag(w2))
    z = z - nm('2')*w2;
end
while inf(imag(z)) < inf(-imag(w2))
    z = z + nm('2')*w2;
end

%
% default values
%

abstol = 1e-16;

%
% constants
%

tau = w2/w1;
z = z/(nm('2')*w1);

%
% process optional user input
%

alen = length(varargin);
pos = 0;
while pos < alen
    pos = pos+1;
    com = varargin{pos};
    if isa(com,'char')

```

```

switch lower(com)
    % the smallest requirement on truncation error
    case 'abstol'
        pos = pos + 1;
        abstol = vararginpos;
    otherwise
        error(['user input, ',com,' is not an option']);
end

else
    error('User error in optional arguments');
end
end

% auxiliary Jacobi theta functions
v01z = theta_func(z+nm('0.5'),tau);
v11z = exp(nm('0.25')*nm('pi')*nm('1i')*tau+nm('pi')*nm('1i')*(z+nm('0.5')))...
    *theta_func(z+nm('0.5')*tau+nm('0.5'),tau);

v00zzero = theta_func(nm('0'),tau);
v10zzero = exp(nm('0.25')*nm('pi')*nm('1i')*tau)*theta_func(nm('0.5')*tau,tau);

v01z_der = theta_func_der(z+nm('0.5'),tau,'abstol',abstol);
v11z_der = nm('1i')*nm('pi')*v11z+...
    exp(nm('0.25')*nm('pi')*nm('1i')*tau+nm('pi')*nm('1i')*(z+nm('0.5')))...
    *theta_func_der(z+nm('0.5')*tau+nm('0.5'),tau,'AbsTol',abstol);

% constant
A = nm('pi')^nm('2')*v00zzero^nm('2')*v10zzero^nm('2');

% derivative
pz = nm('2')*A*(v01z*v01z_der/v11z^nm('2')-v01z^nm('2')*v11z_der/v11z^nm('3'));

% change of variables
out = pz/(nm('2')*w1)^nm('3');
```

3.56 weierstrass_sigma.m

```
function out = weierstrass_sigma(z,omega_1,omega_2,varargin)

% use periodicity of the Weierstrass sigma function to bring argument
% close to origin where the theta_func program can compute more easily
m1 = 0;
m2 = 0;
while sup(real(z)) > sup(real(omega_1))
    z = z - nm('2')*omega_1;
    m1 = m1 + 1;
end
while inf(real(z)) < -inf(real(omega_1))
    z = z + nm('2')*omega_1;
    m1 = m1 - 1;
end
while sup(imag(z)) > sup(imag(omega_2))
    z = z - nm('2')*omega_2;
    m2 = m2 + 1;
end
while inf(imag(z)) < inf(-imag(omega_2))
    z = z +nm('2')*omega_2;
    m2 = m2 - 1;
end

%
% default values
%

AbsTol = 1e-16;

%
% constants
%

omega_1 = real(omega_1);
tau = omega_2/omega_1;
q = exp(-nm('pi')*imag(tau));
```

```

%
% process optional user input
%

alen = length(varargin);
pos = 0;
while pos < alen
    pos = pos+1;
    com = vararginpos;
    if isa(com,'char')

        switch lower(com)
            % the smallest requirement on truncation error
            case 'abstol'
                pos = pos + 1;
                abstol = vararginpos;
            % Weierstrass eta_1
            case 'eta1'
                pos = pos + 1;
                eta_1 = vararginpos;
            otherwise
                error(['user input, ',com,', is not an option']);
        end

    else
        error('User error in optional arguments');
    end
end

%
% choose k large enough that error is less than AbsTol
%

k = 1;
ks = nm(num2str(k));
% ensure k is large enough that error_bound < AbsTol
c1 = nm('1')/(nm('4')*ks*(nm('1')-q^(nm('2')*ks)));
c2 = exp(-imag(tau)*nm('pi')*nm('2')-imag(z)*nm('pi')/omega_1);
c3 = exp(-imag(tau)*nm('pi')*nm('2')+imag(z)*nm('pi')/omega_1);
c4 = exp(-imag(tau)*nm('pi')*nm('2'));

```

```

error_bound = c1*(c2^ks/(nm('1')-c2)+nm('2')*c4/(nm('1')-c4)+c3^ks/(nm('1')-
c3));
while sup(error_bound) > AbsTol
    k = k+1;
    ks = nm(num2str(k));
    c1 = nm('1')/(nm('4')*ks*(nm('1')-q^(nm('2')*ks)));
    error_bound = c1*(c2^ks/(nm('1')-c2)+nm('2')*q^(nm('2')*ks)/(nm('1')-
c4)+c3^ks/(nm('1')-c3));
end

sum_error_bound = hull(-abs(error_bound),abs(error_bound))...
    +nm('1i')*hull(-abs(error_bound),abs(error_bound));

sum = nm('0');
for n = 1:k-1
    ns = nm(num2str(n));
    sum = sum + (q^(nm('2')*ns)/(ns*(nm('1')-...
    q^(nm('2')*ns))))*sin(ns*nm('pi')*z/(nm('2')*omega_1))^nm('2');
end

sum = sum + sum_error_bound;

if (exist('eta_1')==1
    eta_1 = weierstrass_eta1(omega_1,omega_2);
end

lg = eta_1*z^nm('2')/(nm('2')*omega_1)+nm('4')*sum;
out = (nm('2')*omega_1/nm('pi'))*(sin(nm('pi')*z/(nm('2')*omega_1)))*exp(lg);

% use quasi-periodicity of the Weierstrass zeta function
if (m1==0)&&(m2==0)
    return
else
    m1s = nm(num2str(m1));
    m2s = nm(num2str(m2));
    eta_2 = weierstrass_zeta(omega_2,omega_1,omega_2);
    out = out *nm('-1')^(m1s*m2s+m1s+m2s)...
        *exp(nm('2')*(m1s*eta_1+m2s*eta_2)*(z+m1s*omega_1+m2s*omega_2));
end

```

3.57 weierstrass_zeta.m

```
function out = weierstrass_zeta(z,omega_1,omega_2,varargin)

% use periodicity of the Weierstrass zeta function to bring argument
% close to origin where the theta_func program can successfully compute
m = 0;
n = 0;
while sup(real(z)) > sup(real(omega_1))
    z = z - nm('2')*omega_1;
    m = m + 1;
end
while inf(real(z)) < inf(-real(omega_1))
    z = z + nm('2')*omega_1;
```

```

        m = m - 1;
    end
    while sup(imag(z)) > sup(imag(omega_2))
        z = z - nm('2')*omega_2;
        n = n + 1;
    end
    while inf(imag(z)) < inf(-imag(omega_2))
        z = z + nm('2')*omega_2;
        n = n - 1;
    end

    tau = omega_2/omega_1;
    % nome
    q = exp(nm('pi')*nm('li')*tau);

    %
    % user error check
    %

    % make sure tau lies in the upper half of complex plane
    if imag(tau) <= 0
        error('omega_2/omega_1 must lie in the upper half of the complex plane')
    end

    % make sure tau is purely imaginary
    if real(tau) > 0
        error('tau must be purely imaginary');
    end

    %
    % default values
    %

    % absolute error tolerance of truncated remainder
    abstol = 1e-16;

    %
    % process optional user input
    %

```

```

alen = length(varargin);
pos = 0;
while pos < alen
    pos = pos+1;
    com = vararginpos;
    if isa(com,'char')

        switch lower(com)
            % the smallest requirement on truncation error
            case 'abstol'
                pos = pos + 1;
                abstol = vararginpos;
            % Weierstrass eta_1
            case 'eta1'
                pos = pos + 1;
                eta_1 = vararginpos;
            otherwise
                error(['user input, ',com,', is not an option']);
        end

    else
        error('User error in optional arguments');
    end
end

if (exist('eta_1'))==1
    eta_1 = weierstrass_eta1(omega_1,omega_2);
end

% k so that the error bound is less than AbsTol
k = 1;
error_bound = nm(abstol + 1);
a_plus = (nm('pi')/omega_1)*( -nm('2')*imag(omega_2)+abs(imag(z)));
a_minus = (nm('pi')/omega_1)*( -nm('2')*imag(omega_2)-abs(imag(z)));
con_plus = nm('1')-exp(a_plus);
con_minus = nm('1')-exp(a_minus);
cnt = 0;
while sup(error_bound) > abstol
    cnt = cnt + 1;
    if cnt > 1000
        error('failure to find error bound');
    end
end

```



```

    k = k+1;
    ks = nm(num2str(k));
    c = (nm('0.5')/(nm('1')-q^(nm('2')*ks)));
    error_bound = abs((c*(exp(a_plus)^ks/con_plus + exp(a_minus)^ks/con_minus)));
end

% add the first k-1 terms of the sum
sum = nm('0');
for j = 1:k-1
    js = nm(num2str(j));
    sum = sum + (q^(nm('2')*js)/(nm('1')-q^(nm('2')*js)))*sin(js*nm('pi')*z/omega_1);
end

sum = sum + hull(-error_bound,error_bound) + nm('1i')*hull(-error_bound,error_bound);

out = eta_1*z/omega_1+(nm('pi')/(nm('2')*omega_1))*cot(nm('pi')*z/(nm('2')*omega_1))+...
      (nm('2')*nm('pi')/omega_1)*sum;

% use quasi-periodicity of the Weierstrass zeta function
if (m==0)&&(n==0)
    return
else
    out = out + nm('2')*nm(num2str(m))*eta_1+nm('2')*nm(num2str(n))...
            *weierstrass_zeta(omega_2,omega_1,omega_2);
end
end

```

3.58 xi_der_q_psi.m

```

function out = xi_der_q_psi(q,psi,ntilde)
% function out = xi_der_q_psi(q,psi,ntilde)
%
% returns the derivative of omega*xi with respect to psi

```

Now

$$\begin{aligned}\frac{\partial}{\partial \psi} \omega \xi(\omega + i\psi \omega') &= 2\omega \omega' \left(\left(\frac{\pi}{2\omega} \right)^2 \sec^2 \left(\frac{i\pi \psi \omega'}{2\omega} \right) - \frac{\pi^2}{\omega^2} \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} (q^{\psi k} + q^{-\psi k}) \right) \\ &= \frac{-\pi \log(q)}{2} \sec^2 \left(\frac{\psi \log(q)i}{2} \right) + 2\pi \log(q) \sum_{k=1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} (q^{\psi k} + q^{-\psi k})\end{aligned}$$

Note that

$$\begin{aligned}\left| \sum_{k=N+1}^{\infty} (-1)^k \frac{kq^{2k}}{1-q^{2k}} (q^{\psi k} + q^{-\psi k}) \right| &\leq \sum_{k=N+1}^{\infty} \frac{kq^{2k}}{1-q^{2k}} (q^{\psi k} + q^{-\psi k}) \\ &\leq \frac{2}{1-q^{2(N+1)}} \sum_{k=0}^{\infty} (N+1+k) q^{(2-\psi)(N+1+k)} \\ &\leq \frac{2(N+1)q^{(2-\psi)(N+1)}}{1-q^{2(N+1)}} \sum_{k=0}^{\infty} q^{(2-\psi)k} + \frac{2q^{(2-\psi)N}}{1-q^{2(N+1)}} \sum_{k=0}^{\infty} kq^{(2-\psi)k} \\ &\leq \frac{2(N+1)q^{(2-\psi)(N+1)}}{1-q^{2(N+1)}} \frac{1}{1-q^{(2-\psi)}} + \frac{2q^{(2-\psi)N}}{1-q^{2(N+1)}} \frac{q^{(2-\psi)}}{(1-q^{(2-\psi)})^2}\end{aligned}$$

%%

%
% constants
%

```
pie = nm('pi');
psi0 = max(sup(psi));
q0 = abs(q^(2-pi0));
qs0 = q0^(2-sup(abs(psi0)));
q02 = q0*q0;
```

%
% find N for given error
%

```

N = 0;
maxit = 1000;
tol = 1e-17;
err = tol + 1;
while err > tol
    N = N+1;
    if N > maxit
        error('Maximum iterations exceeded');
    end
    temp = 2*(N+1)*qs0^(N+1)/((1-q02^(N+1))*(1-qs0)) + 2*qs0^(N+1)/((1-
q02^(N+1))*(1-qs0)^2);
    err = sup(temp);
end

% initialize output
out = nm(zeros(1,length(psi)));

% find partial sum
q2 = q*q;
for k = 1:N
    q2k = q2^k;
    out = out + ((-1)^ntilde)^k*(k*q2k/(1-q2k))*(q.^(k*psi)+q.^(-k*psi));
end

% add error
out = out + nm(-err,err)+1i*nm(-err,err);

% add non infinite sum parts
if ntilde == 0
    out = (-pie*log(q)/2)./(sin(1i*log(q)*psi/2).^2) + 2*pie*log(q)*out;
elseif ntilde == 1
    out = (-pie*log(q)/2)./(cos(1i*log(q)*psi/2).^2) + 2*pie*log(q)*out;
end

```

3.59 xi_q_psi.m

```
function out = xi_q_psi(q,psi,ntilde)
% computes  $\omega \cdot \xi(\tilde{n}\omega + i\psi\omega_{\text{prime}})$ 
```

From

$$\vartheta_1(z) = 2 \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} \sin((2n-1)z),$$

we find that

$$\begin{aligned} f(x) &:= \vartheta_1 \left(\frac{\pi}{2\omega} (\omega x + i\omega' + \tilde{n}\omega + i\psi\omega') \right) \\ &= -i \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n+1)} \right), \end{aligned}$$

where $\hat{v} := e^{i\pi(x+\tilde{n})/2} q^{(1+\psi)/2}$. Hence

$$\frac{\partial^m}{\partial x^m} f(x) = -i(i\pi/2)^m \sum_{n=1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} (2n-1) \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n+1)} \right).$$

We find that

$$\begin{aligned} Err &:= \left| -i(i\pi/2)^m \sum_{n=N+1}^{\infty} (-1)^{n+1} q^{(n-1/2)^2} (2n-1) \left(\hat{v}^{(2n-1)} - \hat{v}^{-(2n+1)} \right) \right| \\ &\leq 2q^{1/4} (\pi/2)^m \sum_{n=N+1}^{\infty} q^{n^2/2} \left((2n-1)^m q^{n^2/2 - n - (2n-1)(1+\psi)/2} \right) \\ &\leq 2q^{1/4} (\pi/2)^m \sum_{n=N+1}^{\infty} q^{n^2/2} \\ &\leq 2q^{1/4} (\pi/2)^m q^{(N+1)^2/2} \frac{1}{1-q}, \end{aligned}$$

as long as we take N large enough so that $g(x) := \left((2x-1)^m q^{n^2/2 - x - (2x-1)(1+\psi)/2} \right)$ satisfies $g(N) < 1$, $g(x) < 0$ whenever $x > N$.

%%

%
% constants
%

```

pie = nm('pi');
psi0 = max(sup(psi));
q0 = abs(q^(2-psi0));

%
% find N for given error
%

N = 0;
maxit = 1000;
tol = 1e-17;
err = tol + 1;
while err > tol
    N = N+1;
    if N > maxit
        error('Maximum iterations exceeded');
    end
    temp = q0^(N+1)/((1-q^(2*(N+1)))*(1-q0));
    err = sup(temp);
end

% initialize output
out = nm(zeros(1,length(psi)));

% add partial sum
q2 = q*q;
logq = log(q);
for k = 1:N
    q2k = q2^k;
    out = out + (q2k/(1-q2k))*sin(k*(pie*ntilde-li*psi*logq));
end

% add error
out = out + nm(-err,err) + 1i*nm(-err,err);

% multiply by constants
out = 2*pi*out;

```

```

% add non infinite sum parts
out = out + (pie/2)*cot(pie*ntilde/2-1i*psi*logq/2);

% multiply by constants to get omega*xi
out = 2*1i*out;

```